UNIVERSIDAD
REY JUAN CARLOS

INGENIERÍA INFORMÁTICA

Curso académico 2004-2005

Proyecto Fin de Carrera

SISTEMA DE ATENCIÓN VISUAL EN ESCENA

Autora: Marta Martínez de la Casa Puebla

Tutor: Jose María Cañas Plaza

Para mi familia y mis amigos.

Agradecimentos.

Quisiera agradecer a José Mª Cañas su paciencia y el haberme facilitado la ayuda y los conocimientos necesarios de robótica para la realización de este proyecto.

De igual modo, agradecer también a mi familia, que siempre han estado ahí apoyándome y que gracias a ellos he podido hacer lo que siempre me ha gustado.

También agradecer a mi novio Jorge, su paciencia y el saber escucharme en los momentos malos que he tenido a lo largo de estos dos años en los que llevamos juntos, cosa que sé que no es nada fácil. De igual modo, agradecerle también todos los buenos momentos que hemos pasado y los que aún nos quedan.

A mis amigos de siempre, Bea, Luis, Marina, Patricia, Pili, por haber estado ahí todos estos años, dándome su apoyo.

A Juan Ignacio, por todos estos buenos momentos en la Universidad en los que nos hemos reído juntos.

Por último a mis compañeros de universidad, los cuáles me han hecho pasar de una forma llevadera estos 6 años de carrera, los cuáles han sido de los mejores de mi vida.

A todos, Gracias.

Índice general

Índice de figuras

Índice de cuadros

Resumen

Los sistemas de visión son hoy en día uno de los elementos sensoriales más atractivos para incrementar la autonomía en robótica. Las cámaras proveen mucha información sobre el entorno del robot y son unos sensores bastantes baratos. Pero tiene una desventaja, y es que extraer información a apartir de las imágenes capturadas por una cámara es algo difícil, en parte porque hay que distinguir aquello que es importante dentro de las imágenes de lo que no lo es. Para ello están los sistemas de atención visual.

El presente proyecto realiza un algoritmo de atención visual en una escena cuyo campo de visión abarca más que el propio de una sóla cámara. Para ello mantiene una representación de la escena que rodea a un robot y realiza un control de atención sobre los objetos relevantes que aparecen en esa escena. Como la escena abarca mucho más de lo que contienen las imágenes de una cámara si permanece fija, se ha empleado un cuello mecánico que se sitúa sobre el robot.

El algoritmo mueve de manera inteligente este cuello con el objetivo de encontrar objetos nuevos, seguir a los ya existentes y olvidar a aquellos objetos que ya no aparezcan en la escena. Este algoritmo diseñado e implementado, se basa en dos dinámicas concurrentes, la vida y la saliencia. La vida aumenta cuando se observa el objeto y disminuye cuando no es observado. La saliencia en cambio aumenta cuando no es observado y se pone a cero cuando sí lo es. Con estas dos dinámicas tan simples se ha logrado las funcionalidades mencionadas.

Para su implementación, se ha empleado la plataforma software jde.c y se han programado una serie de esquemas escritos en C. Concretamente se han programado dos esquemas, uno que implementa el algoritmo de atención visual y la representación de la escena; y otro que ayuda a la visualización y depuración de este primero.

Capítulo 1

Introducción

En este capítulo se va a dar una visión panorámica del contexto del presente proyecto. Para ello se comienza dando una visión general del mismo, describiendo que es la robótica y la visión artificial en robots. Finalmente, se hace incapié finalmente en el contexto más próximo en el que se centra, que son los sistemas de atención visual.

1.1. Robótica

La palabra robot deriva de la palabra checoslovaca *robota*, que significa trabajador, sirviente. Surge en la obra RUR, los *"Robots Universales de Rossum"*, escrita por Karel Capek. En esta obra se plantea la construcción de robots para liberar a las personas de la carga pesada del trabajo. A su vez, el escritor Isaac Asimov, a quién se atribuye el término robótica, previó un mundo futuro en el que existían reglas de seguridad para que los robots no pudieran ser dañinos para los seres humanos. Siguiendo en esta línea siempre han existido novelas de ciencia-ficción en las que se creaban seres artificiales a imagen y semejanza de los seres humanos. Ejemplos de estas novelas son *Frankenstein*, la vieja leyenda judía del *Golem*, etc. Pero todo esto se queda en ficción. En la realidad fue en 1954 cuando George Devol desarrolló un brazo primitivo o manipulador que sería el origen real de los robots industriales.

Los robots industriales (figura 1.1) fueron evolucionando y tuvieron su apogeo durante los años 70 y 80. Gracias a ellos se dio lugar a unos procesos de producción mucho más eficientes y aportaron mayor calidad a los productos. Estos robots suelen ser empleados en tareas repetitivas y pesadas, como pueden ser el soldado, el pintado o la carga de maquinaria. Suelen estar pre-programados para esa tarea y no disponen de capacidad para reconfigurarse autónomamente.

Figura 1.1: Ejemplo de un robot industrial

Frente a estos robots, a finales de los años 70 y a principios de los años 80, surgió una nueva aproximación en la robótica en el mundo de la investigación. Este nuevo tipo de robótica se denomina *Robótica Autónoma*. Los robots autónomos son sistemas que operan eficientemente en entornos complejos sin necesidad de estar constantemente guiados y controlados por operadores humanos. Una propiedad fundamental que poseen es la de poder reaccionar adecuadamente ante situaciones no previstas explícitamente en su programación previa. Por ello, se suele considerar a la robótica autónoma un campo muy relacionado con la inteligencia artificial, puesto que para generar un comportamiento adecuado, se requiere de una cierta inteligencia mínima.

Actualmente, los robots tienen diversas aplicaciones. Una muy importante es el uso que se les da en tareas peligrosas o desagradables para los seres humanos, como pueden ser exploraciones submarinas, volcánicas o en exploraciones espaciales. Ejemplo de ellas la realizada por la sonda espacial Opportunity de la NASA (figura 1.2), que viajó a Marte y realizó tareas como la detección de agua subterránea en dicho planeta.

Actualmente se ha logrado un gran avance en los robots dedicados a la medicina, y es que se han empleado robots en procedimientos de cirugía ya que la precisión de movimiento de un robot es superior a la de las manos de un cirujano, inevitablemente sujetas a movimientos no deseados o a errores de posicionamiento por falta de

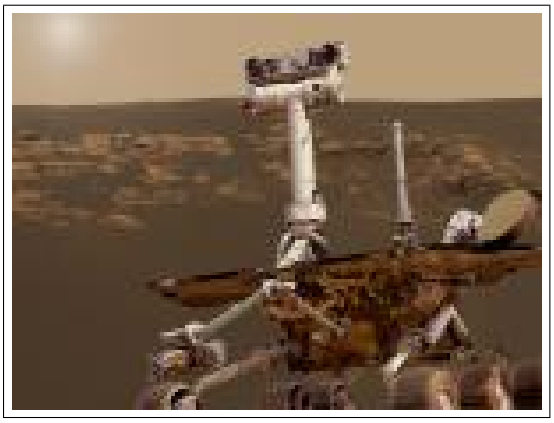


Figura 1.2: Sonda espacial Opportunity

visibilidad, cansancio, etc. También, se usan robots dentro de los laboratorios para transportar muestras biológicas o químicas entre instrumentos tales como incubadoras, manejadores de líquidos y lectores.

La otra aplicación a destacar es la robótica de servicio. Los robots pueden colaborar dentro de nuestras casa y oficinas en multitud de tareas, que van desde el simple ocio (robots mascotas, como el perrito Aibo de Sony (figura 1.3)), pasando por el desarrollo de tareas de funciones domésticas (aspiradora Roomba(figura 1.3), corte de césped), hasta tareas de vigilancia y seguridad. Estos robots están experimentando un grado de aceptación creciente, estando aún limitada su comercialización debido al coste todavía elevado.

Figura 1.3: El robot aibo de sony (Izquierda) y el robot aspiradora roomba (Derecha).

1.2. Visión Artificial en Robótica

La visión artificial, o visión por computador, es una rama de la inteligencia artificial que tiene por objetivo analizar y proponer herramientas para el procesamiento de imágenes, reconocimiento de patrones (como pueden ser caras), reconstrucción de mapas 3D, etc . Además, es una ciencia que posee numerosas aplicaciones en el mundo real, como puede ser dentro de la medicina, la industria, la robótica, etc.

La visión artificial surge en la década de los 60 con la idea básica de conectar una cámara de vídeo a una computadora. Esto implicó no sólo la captura de imágenes a través de la cámara, sino también la comprensión de lo que estas imágenes representaban. Un resultado muy importante de este trabajo y que marcó el inicio de la visión artificial, fue un trabajo realizado por Larry Roberts, el creador de *ARPAnet*. En 1961 creó un programa, el *"mundo de micro-bloques"*, en el que un robot podía "ver" una estructura de bloques sobre una mesa, analizar su contenido y reproducirla desde otra perspectiva, demostrando así que esa información visual que había sido mandada al ordenador por una cámara, había sido procesada adecuadamente por él. Sus herramientas están agrupadas por el procesamiento digital de imágenes y por el reconocimiento de patrones.

Desde entonces la visión artificial ha ido evolucionado y hoy en día uno de los elementos sensoriales más atractivos para incrementar la autonomía en robótica, son los sistemas de visión. Éstos proveen información relevante sobre el estado de los robots y de su entorno físico inmediato y es un sensor bastante barato en comparación con otros, por ejemplo los sónares. La única dificultad que puede existir al emplear una cámara como sensor principal es la extracción de información a partir de una imagen. Ésta debería ser analizada para extraer información relevante, lo cuál haría que fuera algo más complejo que el extraer información de otros tipos de sensores, como por ejemplo los sónares.

Dentro de la robótica la visión artificial tiene diversos usos. Como puede ser en el reconocimiento de patrones. Las cámaras que poseen los robots van tomando imágenes que serán analizadas para identificar objetos de interés, para luego posteriormente realizar alguna acción sobre ellos.

Otro uso es la localización. El robot emplea la visión para saber en donde se

encuentra y en qué orientación está dentro de su entorno. Ejemplos de este uso están en los proyectos *Localización probabilística en un robot con visión local* [Crespo03] y *Localización visual del robot Pioneer* [Lopez05], proyectos realizados dentro del grupo de robótica de la URJC [1].

Un posible uso más sería la navegación para robots. Gracias a la visión los robots pueden crear mapas del entorno en el que se desea que se desplacen. Con esta información el robot podrá ir de un punto a otro dentro de un escenario de manera autónoma. Ejemplos de este empleo está en los proyectos *Comportamiento sigue pelota en un robot con visión local* [SanMartin02], *Comportamiento sigue pared en un robot con visión local* [Gomez02] o el *Comportamiento sigue pelota con visión cenital* [Martinez03].

1.3. El problema de la Atención Visual

Dentro de la visión artificial se encuentra la atención visual. La atención es la fijación en uno o varios aspectos de la realidad y prescindir de los restantes. Esto es, tener preferencia sobre ciertos ojetos que sobresalen por alguna característica, ya sea por su color, forma, etc. La atención dispone de dos etapas claramente marcadas, la primera, considerada procesamiento previo, es aquella en la que se extraen objetos que cumplen determinadas características, dentro del campo visual. Y la segunda, llamada atención enfocada, consiste en la identificación de esos objetos.

Dentro de la robótica autónoma es importante el realizar un control de atención visual. Las cámaras de los robots proveen de un amplio flujo de datos del que hay que seleccionar lo que es interesante e ignorar lo que no, en esto consiste la atención visual selectiva. Existen dos vertientes de atención visual, *global* (*overt attention*) y *local* (*covert attention*). La atención *local* consiste en seleccionar de una sóla imagen aquellos datos que nos interesan. Y la atención *global* consiste en seleccionar del entorno que rodea al robot aquellos objetos que interesan, a los que dirigir la mirada [Cañas05].

Siguiendo la línea de la atención global selectiva se presenta el presente proyecto. Este proyecto está enfocado a realizar un control de atención sobre diferentes objetos situados en una escena visual, cuyo campo de visión abarca más que el propio

[1] http://gsyc.escet.urjc.es/robotica

de una sóla cámara. Los objetos a atender serán relevantes por su color, en especial se desea atender objetos rosas y azules. Se tiene que poder identificar estos objetos, realizar un seguimiento entre ellos, olvidarlos cuando desaparezcan de la escena e incluso priorizar su atención según su color. Para su realización se dispone de la plataforma de la figura 1.4. En ella se tiene un robot que posee un cuello mecánico con una cámara. Esto permitirá tener una representación del entorno más amplio que el propio campo de visión de una cámara.

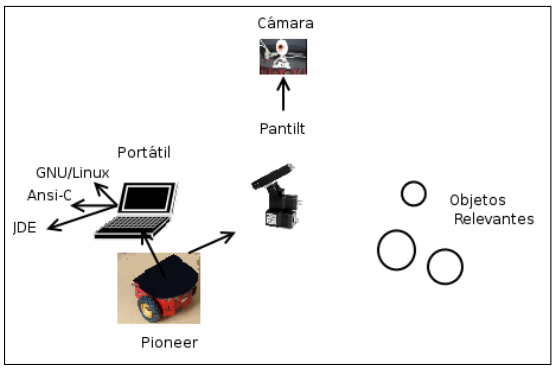


Figura 1.4: Plataforma empleada para el sistema

También se ha dispuesto de la plataforma *jde.c* [Cañas03], plataforma empleada en el grupo de robótica de la URJC[2]. Siguiendo esta plataforma se están implementado todos los proyecto de dicho grupo, pero este es el primero que aborda el problema de la atención visual.

Esta memoria consta de seis capítulos. En el segundo capítulo se describirá cuál es el objetivo concreto de este proyecto y qué requisitos, funcionales y no funcionales, se deben cumplir para un correcto funcionamiento del mismo. En el tercero se describirán las herramientas necesarias para su implementación, tanto herramientas hardware como software. En el capítulo cuatro se explicará cómo se ha desarrollado el sistema, hablando tanto del diseño como de la implementación del mismo. En el capítulo cinco se comentarán las pruebas realizadas para observar cómo se han cumplido los objetivos y ver el correcto funcionamiento del sistema. Finalmente, se presentará en el capítulo seis las conclusiones que se han obtenido, y qué mejoras se pueden efectúar para posibles trabajos futuros.

[2] http://gsyc.escet.urjc.es/robotica

Capítulo 2

Objetivos

Una vez explicado en el capítulo anterior el contexto en el que se sitúa el siguiente proyecto, en el siguiente capítulo se presenta una descripción de lo que se quiere que resuelva el sistema. Para ello se definirá de modo preciso el problema que se quiere resolver con este proyecto fijando el objetivo principal y los subobjetivos en los que se descompone, la funcionalidad requerida, las restricciones que posee.

2.1. Descripción del Problema

El objetivo principal de este proyecto consiste en conseguir realizar un seguimiento sobre unos objetos que son relevantes en una escena más amplia que el campo visual de una sóla cámara. Para realizar este seguimiento, se debe captar nuevos objetos, alternar la atención entre los existentes, olvidarlos una vez que hayan desaparecido, seguir su movimiento, manejar prioridades entre los objetos a seguir, realizar una búsqueda de nuevos objetos, etc. Para ello se va a disponer de una cámara digital situada sobre un cuello mecánico. Esto es así porque el campo de visión de la misma es limitado, por lo que hará uso del cuello mecánico para llegar a un mayor número de puntos.

Este objetivo se ha dividido a su vez en tres subobjetivos:

1. ***"Percepción Monocular"*:** La cámara deberá ir capturando imágenes continuamente para conocer la información del entorno en el que está situada. Estas imágenes serán analizadas mediante técnicas propias de visión computacional, filtrado y la segmentación, para identificar los objetos relevantes.

2. ***"Representación en Escena"*:** Se desea tener una representación visual de la escena. Para ello se sitúa la cámara en un cuello mecánico, ya que de este modo se logrará un mayor ángulo de visión. El cuello se moverá en diferentes ángulos para que de ese modo la imagen de escena abarque todo el entorno próximo del robot. Cada imagen tomada por la cámara, servirá para formar la imagen de escena, porque ésta última está compuesta por cada una de ellas.

3. ***"Control de Atención"*:** Se deberá realizar un control de atención sobre los diferentes objetos identificados. Para ello se deberá captar nuevos objetos, alternar entre los objetos existentes, olvidar los objetos cuando desaparezcan, seguir su movimiento, manejar prioridades entre los objetos a seguir, etc.

2.2. Especificación de Requisitos

En la siguiente sección se presentan los requisitos que deberán ser satisfechos por el sistema. Todos los requisitos aquí expuestos son **ensenciales**, es decir, no sería aceptable un sistema que no satisfaga alguno de los requisitos aquí presentados.

2.2.1. Requisitos Funcionales

La funcionalidad que se espera del sistema es la siguiente:

- **Requisito(01)** ***"Capturar Imágenes"*:** El sistema deberá ser capaz de realizar una toma de imágenes mediante una cámara digital para poder disponer de información del entorno en el que se encuentra.

- **Requisito(02)** ***"Distinguir objetos relevantes de los no relevantes en una imagen"*:** Las imágenes capturadas deberán ser tratadas para distinguir objetos relevantes sobre los que se tiene que realizar un seguimiento.

- **Requisito(03)** ***"Reconocer nuevos objetos"*:** Cuando se analice una imagen y se descubra un objeto no identificado previamente, se deberá incorporar para volver a ser visitado.

- **Requisito(04)** ***"Realizar un seguimiento sobre los diferentes objetos encontrados"*:** Los diferentes objetos encontrados deberán ser atendidos periódicamente, es decir, deberán ser visitados de vez en cuando para poder realizar un seguimiento sobre ellos.

- **Requisito(05)** ***"Priorizar objetos"*:** Se deberá poder poner prioridades a los objetos en cuánto a la atención que deben recibir. Esto es, cuanto más prioridad tenga un objeto, más frecuentemente deberá ser visitado.
- **Requisito(06)** ***"Olvidar objetos desaparecidos"*:** Cuando un objeto previamente existente desaparezca de la escena, deberá ser olvidado.
- **Requisito(07)** ***"Explorar zonas en busca de objetos"*:**Se deberá explorar zonas de la escena con la idea de captar nuevos objetos.
- **Requisito(08)** ***"Componer la Imagen de Escena"*:** A partir de las imágenes monoculares obtenidas por la cámara, se ha de crear una imagen que represente la escena. Esta imagen se tiene que poder refrescar continuamente.
- **Requisito(09)** ***"Visualizar la Imagen de Escena"*:** Se tendrá que poder visualizar la imagen del entorno mediante una interfaz y depurar el sistema de atención.

2.2.2. Requisitos no funcionales

Una vez detallada la funcionalidad del sistema, se pasa a citar las restricciones del sistema final.

Requisitos del Producto:

- El sistema deberá ser lo más robusto posible, especialmente frente a cambios en la iluminación. Ha de funcionar correctamente en distintos entornos con diferentes condiciones físicas sin que se tenga que variar nada de su implementación.
- La captura de imágenes y su posterior tratamiento ha de ser en tiempo real, por ello se deberá implementar algoritmos rápidos y eficientes.

Requisitos del Proceso:

- El lenguaje de programación empleado para la implementación del sistema será **ansi-C.**
- El sistema operativo ultilizado será GNU/Linux, puesto que es el usado en el laboratorio de robótica.

- El sistema será diseñado de acuerdo a la arquitectura JDE y su implementación software actual *jde.c*. Se utilizará esta arquitectura porque, además de que es la propuesta por el grupo de robótica, soluciona muchos problemas como la captura de imágenes, la configuración de los drivers, etc.. Por lo tanto, siguiendo esta arquitectura, se dividirá la solución del problema en pequeños esquemas.

Requisitos Externos:

- El código implementado será software libre, por lo que se podrá utilizar y/o modificar de acuerdo con la licencia GPL (General Public License), publicada por la Fundación de Software Libre.

2.3. Metodología

Para la realización de este proyecto se ha optado por seguir un desarrollo incremental (Figura 2.1). Según este modelo el desarrollo del sistema comienza con una especificación de requisitos bien definidos. Tras ello el sistema se descompone en una serie de incrementos. En cada uno de ellos se desarrolla una parte de la funcionalidad definida, se validará y se efectuará una entrega al cliente.

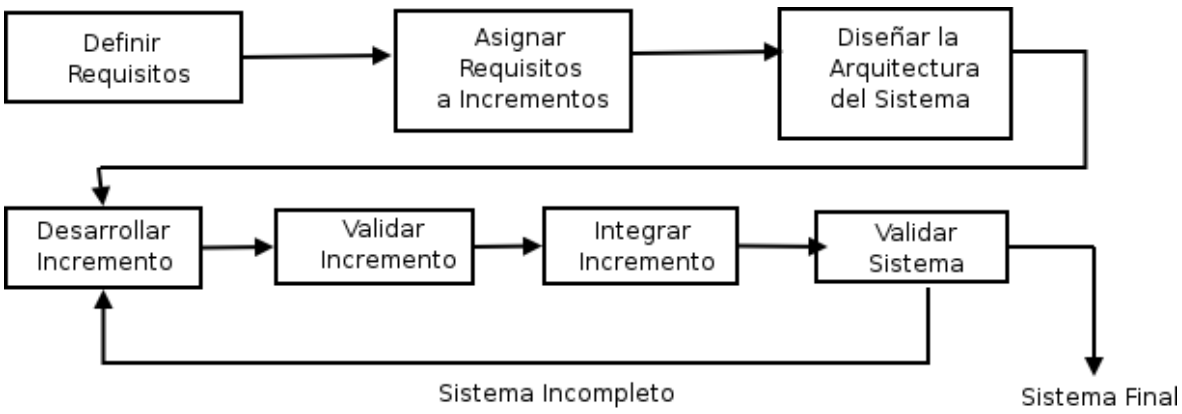


Figura 2.1: Desarrollo Incremental

En el caso del presente proyecto, se ha comenzado por la especificación de requisitos descrita en la sección anterior. Tras ello, los incrementos que se han realizado son los siguientes:

1. Captura de imágenes y tratamiento de las mismas.

2. Composición de la imagen de escena.

3. Desarrollo de la interfaz de usuario en la que se visualiza la imagen de escena.

4. Encontrar objetos relevantes.

5. Realizar la atención sobre esos objetos.

6. Olvidar objetos no observados.

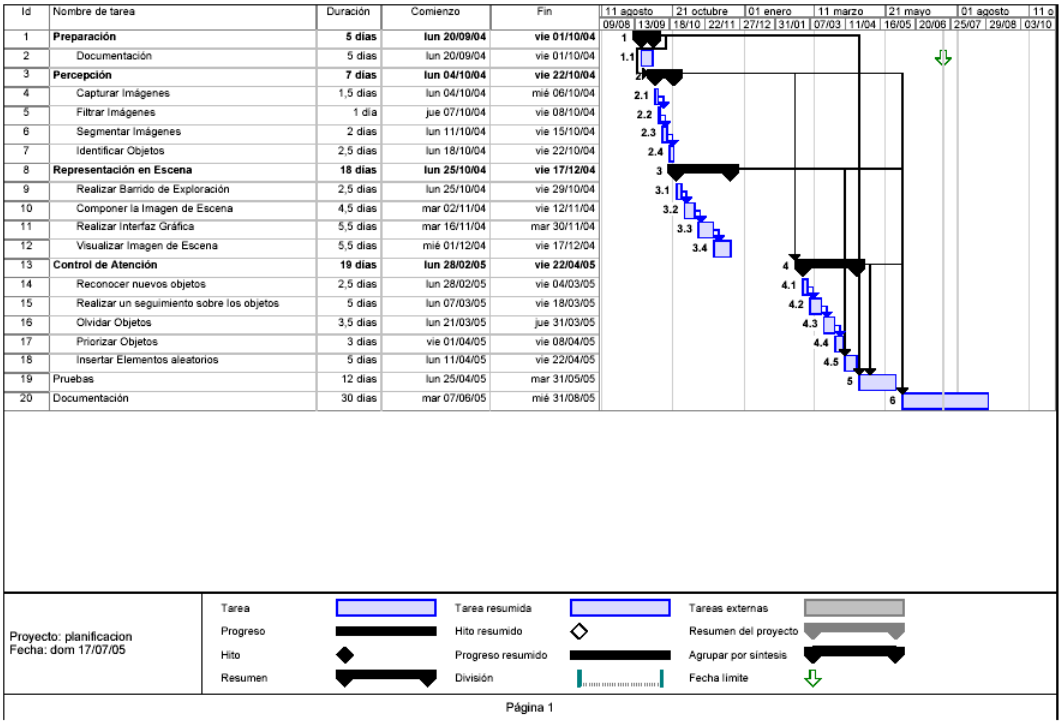
7. Priorizar objetos.

2.4. Planificación del Proyecto

Para finalizar este capítulo se detalla la planificación del proyecto. Para ello, se enumeran las fases principales para la realización del sistema, y las tareas en las que se divide (Tabla 2.1). Además se muestra un diagrama de Gantt en la figura 2.2. En ella quedan reflejadas las tareas y el tiempo estimado para su elaboración.

Tarea	Nombre	Comienzo	Fin	Predecesoras
1	Preparación	lun 20/09/04	vie 01/10/04	
1.1	Documentación	lun 20/09/04	vie 01/10/04	
2	Percepción	lun 04/10/04	vie 22/10/04	
2.1	Capturar Imágenes	lun 04/10/04	mié 06/10/04	
2.2	Filtrar Imágenes	jue 07/10/04	vie 08/10/04	2.1
2.3	Segmentar Imágenes	lun 11/10/04	vie 15/10/04	2.2
2.4	Identificar Objetos	lun 18/10/04	vie 22/10/04	2.2
3	Representación en Escena	lun 25/10/04	vie 17/12/04	2
3.1	Realizar Barrido de Exploración	lun 25/10/04	vie 29/10/04	
3.2	Componer la Imagen de Escena	mar 02/11/04	vie 12/11/04	3.1
3.3	Realizar Interfaz Gráfica	mar 16/11/04	mar 30/11/04	3.2
3.4	Visualizar Imagen de Escena	mié 01/12/04	vie 17/12/04	3.3
4	Control de Atención	lun 28/02/05	vie 22/04/05	2,3
4.1	Reconocer nuevos objetos	lun 28/02/05	vie 04/03/05	
4.2	Realizar un seguimiento sobre los objetos	lun 07/03/05	vie 18/03/05	4.1
4.3	Olvidar Objetos	lun 21/03/05	jue 31/03/05	4.2
4.4	Priorizar Objetos	vie 01/04/05	vie 08/04/05	4.3
4.5	Insertar Elementos Aleatorios	lun 11/04/05	vie 22/05/05	4.4
5	Pruebas	lun 25/04/05	mar 31/05/05	2,3,4
6	Documentación	mar 07/06/05	mié 31/08/05	2, 3, 4, 5

Cuadro 2.1: División en tareas del proyecto



Id	Nombre de tarea	Duración	Comienzo	Fin
1	**Preparación**	**5 días**	**lun 20/09/04**	**vie 01/10/04**
2	Documentación	5 días	lun 20/09/04	vie 01/10/04
3	**Percepción**	**7 días**	**lun 04/10/04**	**vie 22/10/04**
4	Capturar Imágenes	1,5 días	lun 04/10/04	mié 06/10/04
5	Filtrar Imágenes	1 día	jue 07/10/04	vie 08/10/04
6	Segmentar Imágenes	2 días	lun 11/10/04	vie 15/10/04
7	Identificar Objetos	2,5 días	lun 18/10/04	vie 22/10/04
8	**Representación en Escena**	**18 días**	**lun 25/10/04**	**vie 17/12/04**
9	Realizar Barrido de Exploración	2,5 días	lun 25/10/04	vie 29/10/04
10	Componer la Imagen de Escena	4,5 días	mar 02/11/04	vie 12/11/04
11	Realizar Interfaz Gráfica	5,5 días	mar 16/11/04	mar 30/11/04
12	Visualizar Imagen de Escena	5,5 días	mié 01/12/04	vie 17/12/04
13	**Control de Atención**	**19 días**	**lun 28/02/05**	**vie 22/04/05**
14	Reconocer nuevos objetos	2,5 días	lun 28/02/05	vie 04/03/05
15	Realizar un seguimiento sobre los objetos	5 días	lun 07/03/05	vie 18/03/05
16	Olvidar Objetos	3,5 días	lun 21/03/05	jue 31/03/05
17	Priorizar Objetos	3 días	vie 01/04/05	vie 08/04/05
18	Insertar Elementos aleatorios	5 días	lun 11/04/05	vie 22/04/05
19	Pruebas	12 días	lun 25/04/05	mar 31/05/05
20	Documentación	30 días	mar 07/06/05	mié 31/08/05

Figura 2.2: Diagrama de Gantt para la planificación del proyecto

Capítulo 3

Plataforma de Desarrollo

Tras haber explicado los objetivos y los requisitos que se han de llevar a cabo, en este capítulo se describe la plataforma de desarrollo que tiene el sistema. Se comenzará dando una visión general de la misma y a continuación se describirá cada una de las partes, como son el robot *Pioneer*, la jerarquía *JDE* y el cuello mecánico *Pan-Tilt*, entre otros.

3.1. Plataforma General

La plataforma planteada para solucionar el problema descrito en el capítulo anterior es la siguiente (Figura 3.1):

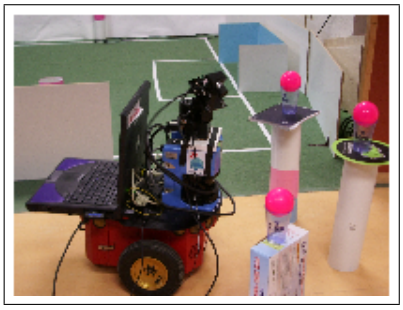


Figura 3.1: Plataforma empleada

Se tiene un robot Pioneer sobre el que está situado un ordenador portátil y un cuello mecánico. En el ordenador portátil está instalado el sistema operativo *GNU/Linux* y sobre él se ejecuta la plataforma *JDE*. Esta jerarquía está basada en esquemas escritos en *C*, por lo que el código está también escrito en dicho lenguaje

de programación. Esta jerarquía soluciona bastantes problemas como la captura de imágenes, configuración de drivers, adaptación de nuestra interfaz, etc.

En cuanto al cuello mecánico, sobre él hay una cámara digital encargada de capturar imágenes. Esta unidad permite un movimiento amplio horizontal y vertical, gracias a ello se puede llegar a un mayor número de zonas en un breve espacio de tiempo.

3.2. El Robot Pioneer

Como se ha explicado, el robot real sobre el que funciona el sistema desarrollado en este proyecto es el robot *Pioneer2DXE* [1] [CañasP04] (Figura 3.2), robot perteneciente a la empresa *ActivMedia*[2].

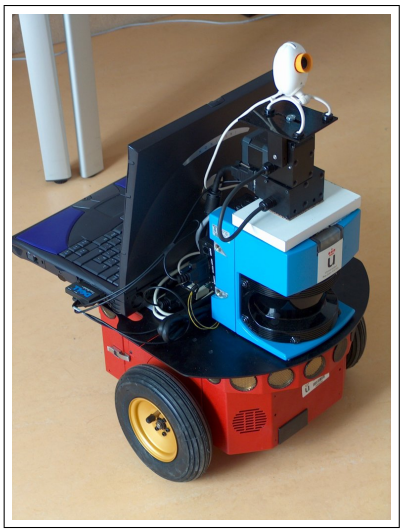


Figura 3.2: Robot Pioneer

Este robot dispone de un equipo de elementos sensoriales para obtener información de su entorno, unos procesadores para realizar cálculos y unos motores para desplazarse por el medio.

[1] http://gsyc.escet.urjc.es/jmplaza/papers/manualjde-2.1c.pdf
[2] http://www.activmedia.com

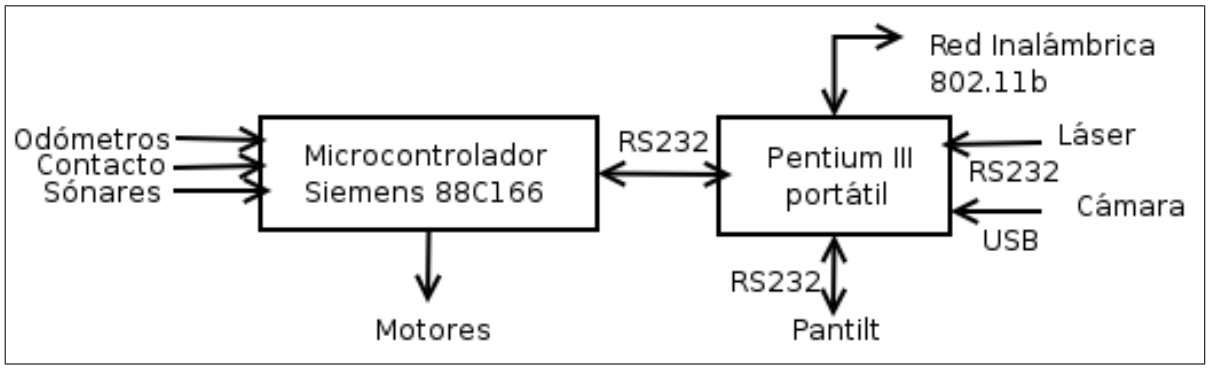


Figura 3.3: Diagrama de Bloques del Hardware del Pioneer

- **Equipo Sensorial:** El robot *Pioneer* dispone de una corona de sensores de contacto en su parte inferior, otra de sensores de ultrasonido y un par de *encoders* asociados a las ruedas motrices. A este equipamiento se le ha añadido un sensor láser y una cámara en la configuración final(Figura 3.3).

- **Procesadores:** El robot dispone de un microporcesador `Siemens 88C166` que ejecuta instrucciones a `20 MHz`. Sobre este microcontrolador funciona un sistema operativo especial llamado `P2OS`, encargado de recoger las medidas de los *sónars*, los *encoders* y de materializar los comandos motores que le llegan. Se conecta a ordenadores externos a través de un puerto serie (Figura 3.3). En este caso se conectará a un ordenador portátil con un procesador `Pentium-III` a `600 MHz`. En este portátil se ejecutarán la mayor parte de los programas para generar el comportamiento.

- **Motores:** Los actuadores principales de los que dispone este robot son dos motores de continua, cada uno asociado a una rueda del *Pioneer*. Con ellos se dota al robot de un movimiento de tracción diferencial (tipo tanque).

Señalar también que el portátil está conectado a la red exterior a través de un enlace inalámbrico, con una tarjeta de red `802.11`, que le proporciona un ancho de banda de `11MBps` en sus comunicaciones hacia el exterior. De este modo el programa de control podrá correr a bordo del robot o en cualquier otro computador.

3.2.1. Cuello Mecánico Pan-Tilt

El cuello mecánico o *Pan-Tilt* (Figura 3.4), como se ha comentado ya, se sitúa sobre el robot *Pioneer*. Se denomina cuello mecánico debido a sus dos movimientos, en horizontal `PAN` y en vertical `TILT`, semejantes a los del cuello humano. Sobre él se apoya

una cámara digital y la proporciona rapidez y exactitud a sus movimientos. Gracias a estos movimientos y a esta precisión, dicha unidad tiene diversas aplicaciones a día de hoy, como pueden ser: uso dentro del campo de la robótica y la visión artificial, utilizarse en control en cámaras de seguridad, emplearse en teleconferencias o incluso en sistemas de monitorización avanzados.

Figura 3.4: Unidad Pan-Tilt o cuello mecánico

En la plataforma de desarrollo usada, el cuello mecánico que se va a emplear es el modelo `PTU-D46-17` (Figura 3.4). Este modelo permite que se conecte a un ordenador mediante el puerto serie `RS-232`. Gracias a esta conexión se le puede enviar comandos del tipo muévete a tal posición, a tal velocidad, etc. El protocolo necesario para realizar esta comunicación viene dada por el fabricante de la misma, y vienen especificados en un manual[3] [Pantilt03]. Ejemplos de los comandos son `PP -2500 *`, muévete a la posición `pan -2500`, `TP500 *` muévete a la posición `tilt 500`, `PS2500 *`, muévete a una velocidad `pan` de `2500 posiciones/segundo` o `TS500 *`, muévete a una velocidad `tilt` de `500 posiciones/segundo`. Debido a que en este sistema se usa la plataforma JDE, este protocolo ya viene resuelto dentro de la misma plataforma, por lo que no hay que preocuparse ni emplear estos comandos de bajo nivel.

Este cuello permite moverse en horizontal con una amplitud de `318°` y de `76°` en vertical. Los movimientos pueden ser absolutos o relativos. En este sistema sólo se emplean los primeros puesto que se le comanda, como se verá en posteriores capítulos, la posición exacta a la que se tiene que mover el cuello.

[3] http://www.dperception.com/pdf/PTU-manual_D46.pdf

Por último comentar que la velocidad máxima capaz de obtener es de `360° por segundo` y con una resolución de `0.514°`. Señalar que la velocidad es modulable y que se puede modificar en todo momento gracias a una interfaz de usuario (sección 4.6).

3.2.2. Cámara

Para obtener información del medio en el que se encuentra el robot se ha de emplear un sensor. El sensor empleado en este proyecto ha sido una cámara. Ésta, como ya se ha comentado, está situada sobre el cuello mecánico. Para la realización de este proyecto se han empleado dos modelos de cámaras: una cámara CCD de sun[4] de videoconferencia conectada a una tarjeta capturadora `BT878` y una cámara firewire [5].

Debido a que la cámara analógica posee menos resolución y transmite imágenes a un ritmo menor que la firewire, dentro de la plataforma de desarrollo descrita al inicio de este capítulo se ha decidido emplear esta segunda. Para las pruebas del sistema, antes de migrar a la plataforma final, se ha empleado la primera.

La cámara empleada en la plataforma final es una cámara digital DCAM-L (Figura 3.5) [6]. Ésta se conecta al PC mediante una conexión firewire y puede transmitir imágenes de una resolución de 640x480 y 320x240, ambas a un ritmo superior de 30 fps. Señalar que en este sistema se emplea una resolución de 320x240.

Las imágenes que puede recoger la cámara pueden estar en un formato en color o en blanco y negro, pero en este sistema están en el primer formato. La cámara tiene una apertura horizontal de 48º y en vertical de 37º.

Un problema añadido al análisis de la imagen es que esta cámara firewire tiene `autoiris`. El `autoiris` es un control electrónico que posee la cámara y que proporciona ajustes automáticos en la imagen para adaptarse a los diferentes niveles de iluminación.

Para recoger las imágenes capturadas por la cámara se ha empleado la plataforma JDE. Esta plataforma será descrita en la sección 3.4.

[4] Sun Camera II, modelo IK-M28SE
[5] DCAM-L IEEE 1394 Digital Color Video Camera
[6] http://www.videredesign.com/dcam.htm

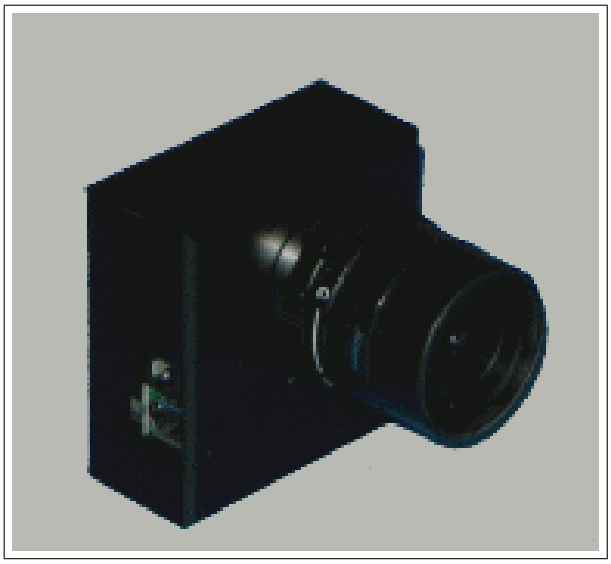


Figura 3.5: Cámara Firewire empleada en el sistema final

3.3. Sistema Operativo y lenguaje de programación

Como se ha comentado, para la implementación del código de este sistema se ha elegido usar el sistema operativo *GNU/Linux* y el lenguaje de programación *C*. A continuación se pasa a describir cada uno de ellos.

GNU/Linux es un sistema operativo similar a *Unix*, de libre distribución, multitarea y multiusuario. Es un sistema de 32 bits, robusto, ágil, muy completo y portado a la mayoría de los procesadores del mercado. La distribución seleccionada dentro de la amplia gama ha sido *Debian*[7] por ser software libre. Por ello está accesible gratuitamente a través de internet y su actualización es muy sencilla.

Por otro lado, una de las desventajas del uso de este sistema operativo es que no es un sistema de tiempo real, ya que no permite acotar plazos porque su sistema de planificación de procesos no implementa esas garantías. Sin embargo, *GNU/Linux* resulta suficientemente ágil para los requisitos necesarios en este proyecto.

En cuanto al lenguaje de programación, como ya se ha mencionado, se ha empleado *C*, ya que la plataforma usada para la elaboración de este proyecto tiene soporte para él. Además, al existir una comunidad muy amplia de grupos de investigación que realizan sus desarrollos en este lenguaje, la reutilización o integración de software se vuelve más factible.

[7] http://www.debian.org

3.4. Plataforma Software JDE y jde.c

Conseguir que los robots hagan cosas útiles autónomamente es una tarea difícil. Unos de los aspectos principales para esta autonomía es la programación que se le da al robot. Todos los robots tienen sensores, actuadores y procesadores, y el software que se ejecuta en ellos es el que enlaza los datos recibidos por los sensores con las respuestas de actuación. Desde esta perspectiva, la generación de comportamiento en un robot consiste en escribir el programa que al ejecutarse en el robot causa este comportamiento.

La manera en la que se programa un robot ha ido evolucionando y gracias al asentamiento de los fabricantes y al trabajo de muchos grupos de investigación han ido apareciendo plataformas de desarrollo que simplifican la programación de aplicaciones robóticas. Estas plataformas ofrecen un acceso más sencillo a sensores y actuadores, y suelen incluir un modelo de programación que establece una determinada organización del software. El diseñador programa sus aplicaciones robóticas sobre esa plataforma de desarrollo (figura 3.6).

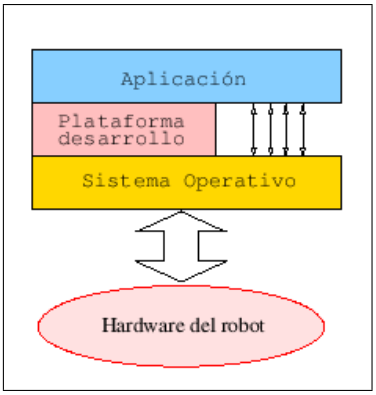


Figura 3.6: Programación de robots sobre una plataforma de desarrollo

La plataforma elegida en este proyecto es la ***J**erarquía **D**inámica de **E**squemas* (***JDE***)[Cañas03][8]. Siguiendo la línea anterior es un entorno para la programación de robots móviles y ha sido diseñada por el Grupo de Robótica de la URJC. Este entorno, al igual que otras plataformas de desarrollo en programación de robots, facilita la creación de programas para que el robot se comporte de una determinada manera y exhiba conductas autónomas. Para ello, resuelve los aspectos más generales de la

[8]http://gsyc.escet.urjc.es/jmplaza/software.html

programación de robots, como el acceso a los sensores y actuadores, la multitarea, las interfaces gráficas y las comunicaciones entre programas. En la práctica incluye tres servidores, varias librerías y un conjunto de ejemplos cuyo código puede servir de base para nuevas aplicaciones.

3.4.1. Arquitectura JDE para plataformas robóticas

La arquitectura JDE se basa en pequeñas unidades de comportamiento denominadas esquemas. Estos esquemas son un flujo de ejecución (modulable, iterativo y puede ser activado o desactivado a voluntad) independiente con un determinado objetivo. Por ello en cada uno de ellos se encapsula diferente funcionalidad que podrá ser reutilizada en posteriores ocasiones.

Existen dos tipos de esquemas en JDE, los esquemas perceptivos y los esquemas motores o de actuación. Los esquemas perceptivos son los encargados de recoger los datos sensoriales y poner esa información a disposición de cualquier esquema. Los esquemas motores utilizan la información recogida por los esquemas perceptivos y generarán una orden de movimiento sobre el robot.

Por último, los esquemas se organizan en una jerarquía dinámica. De este modo, cada esquema puede aprovechar la funcionalidad de otros y materializa así la suya propia. Para ello un esquema activará a otro para usar sus servicios o lo modulará para que se ejecute a un determinado ritmo.

En el caso de este proyecto, no se ha necesitado de los mecanismos jerárquicos, ya que ha bastado con usar un único nivel en donde se integran los diferentes programas creados para generar el sistema.

3.4.2. Plataforma jde.c

La implementación real de la plataforma *JDE* es *jde.c*. Esta implementación propociona un esqueleto para programar los esquemas descritos en la sección anterior, de manera que sólo habría que programar su iteración y aclarar cuáles son sus entradas y cuáles son sus salidas. A su vez, estos esquemas se materializan en una hebra de kernel independiente de las demás. Tanto si el esquema es perceptivo como si es de actuación, cada una de estas hebras ejecuta periódicamente una función de iteración a un ritmo

controlado. Estas hebras se ejecutarán en paralelo con respecto las hebras de los otros esquemas.

La manera en la que cada hebra se comunica es mediante el uso de variables compartidas (tabla 3.1). De este modo se simplica la comunicación entre ellas y se agiliza enormemente en comparación de si se hubiera empleado paso de mensaje entre ellas.

La manera en la que se actualizan esas variables es la siguiente. Un esquema perceptivo leerá un determinado dato sensorial, actualizará una variable común y el resto de esquemas leerán esta variable cuando la necesiten. El uso de estas variables compartidas oculta al programador la complejidad del acceso al hardware. `jde.c` se encargará de la captura de imágenes, configuración de los drivers, etc.

Hay que tener en cuenta que cada vez que se habla de comunciación entre procesos a través de memoria compartida, se habla de condiciones de carrera. En la arquitectura JDE, se producen condiciones de carrera en el acceso a las variables compartidas, pero no se implementan semáforos con el fin de evitar sobrecarga computacional. Estas condiciones de carrera pueden provocar que un esquema motriz envíe una orden errónea de actuación en una determinada iteración, pero al enviar varias de estas órdenes por segundo, este error se podrá corregir en las sucesivas iteraciones, sin que en el movimiento del robot se note la diferencia. De modo que no es imprescindible el uso de semáforos en este caso, y su utilización supondría un mayor coste computacional en la plataforma.

Como se ha dicho, la forma en la que se actualizan las variables es cuando un esquema perceptivo lee un determinado dato sensorial y modifica la variable común. La manera por la que le llegan al esquema perceptivo los diferentes datos sensoriales son tres 3.7. Mediante el robot real, el esquema se comunica directamente al robot y éste le pasa los datos. Esta es la forma más rápida de obtener los datos. Señalar que *jde.c* incluye drivers para manejar localmente las cámaras, accediendo a la interfaz *video4linux* con independencia del tipo de cámara empleada. También, facilita el manejo del cuello mecánico mediante el puerto serie `RS-232`.

Otra forma sería mediante el uso de servidores. Hay dos servidores definidos en

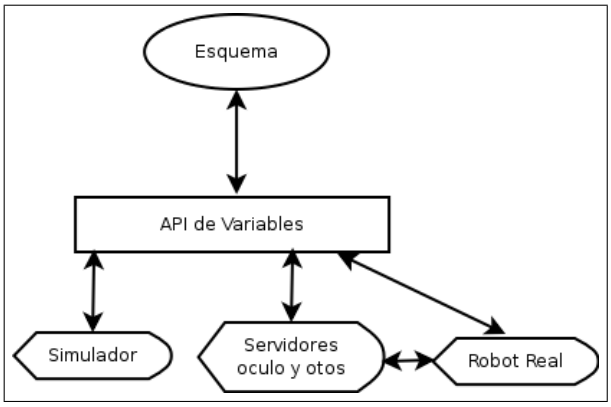


Figura 3.7: Fuentes para la actualización de las variables compartidas

JDE, `otos` y `oculo`. El servidor `otos` reúne los servicios de los sensores de proximidad como los de ultrasonidos, el láser, los infrarrojos y los táctiles. Además, ofrece los datos que generan los odómetros de los motores de tracción del robot. También entrega las medidas del sensor de voltaje de la batería. El otro servidor `oculo` reúne las funciones asociadas a la unidad del cuello mecánico y a la cámara. Con ello permite a los clientes mover la unidad pantilt a voluntad, especificándole ángulos objetivos, y pone a disposición de los clientes el flujo de imágenes obtenidas con la cámara. Además, ambos servidores permiten manejar de manera remota el hardware del robot. Permite, por ejemplo, mover el cuello desde un ordenador diferente al que está conectado. Para este acceso remoto a los sensores y actuadores del robot se dispone de interfaz de mensajes.

La última manera por la que se actualizan las variables sería mediante el empleo de simuladores. Una plataforma empleada por el grupo es *player/stage*. *Stage* es capaz de simular una población de robots móviles, sensores y objetos del entorno. Todos los sensores y actuadores son accesibles a través de la interfaz estándar de *Player*, y usarán también el API de variables descrito.

Cabe señalar que, primeramente en este proyecto se actualizaban las variables empleando los servidores. El servidor usado fue `oculo` ya que en este sistema sólo interesa mover la unidad pantilt y recibir las imágenes de la cámara. Un requisito de este proyecto es la rapidez en la ejecución, y al usar servidores debido al retardo en las comunicaciones se decidió usar el robot real directamente como fuente de la obtención de los datos sensoriales. De este modo se vio, de manera notable, como la ejecución, la recepción de las imágenes y el movimiento del cuello se hacía de una forma más rápida.

El API de variables de `jde.c` para el robot *Pioneer* se puede ver en la tabla 3.1.

Nombre Variable	*Descripción*	*Usada o no*
laser_coord	Contiene las coordenadas de la posición del láser con respecto a la posición de la base del robot.	No
us_coord	Contiene las coordenadas de la posición del sónar con respecto a la posición de la base del robot.	No
camera_coord	Contiene las coordenadas de la posición de la cámara con respecto a la posición de la base del robot.	No
robot	Contiene la posición odométrica del robot.	No
laser	Vector que contiene cada una de las medidas de los puntos del láser. Como mucho habrá 180 puntos.	No
us	Vector que contiene cada una de las medidas de los sónars del robot. Hay 16 sónars.	No
colorA	Variable que contiene una imagen. Ésta puede ser una imagen de un archivo, o una obtenida mediante la cámara izquierda (sistemas con dos cámaras). Esta imagen puede estar en un formato en escala de grises o en color y tiene una resolución de 320x240.	Sí
colorB	Variable que contiene otra imagen. Ésta puede provenir, al igual que la anterior, de un archivo, o de la cámara derecha (sistemas con dos cámaras). Esta imagen puede estar en un formato en escala de grises o en color y tiene una resolución de 320x240.	No
pan_angle	Variable que contiene la posición pan en la que se encuentra el cuello.	Sí
tilt_angle	Variable que contiene la posición tilt en la que se encuentra el cuello.	Sí
longitude	Variable que contiene la posición pan a la que se tiene que desplazar el cuello. Esta posición tiene que estar comprendida en un rango de [-158.º, 158.º].	Sí
latitude	Variable que contiene la posición tilt a la que se tiene que desplazar el cuello. Esta posición tiene que estar comprendida en un rango de [-45.º, 30.º].	Sí
longitude_speed	Variable que contiene la velocidad a la que se tiene que desplazar el cuello en horizontal. Esta velocidad no debe superar los 205.89ºsegundo.	Sí
latitude_speed	Variable que contiene la velocidad a la que se tiene que desplazar el cuello en vertical. Esta velocidad no debe superar los 205.89ºsegundo.	Sí
v	Variable que contiene la velocidad lineal a la que se tiene que desplazar la base del robot. Esta velocidad viene expresada en mmseg. Teniendo un límite de 1000 mm/seg.	No
w	Variable que contiene la velocidad angular a la que se tiene que desplazar la base del robot. Esta velocidad viene expresada en gradosseg. Teniendo un límite de 180º/seg.	No

Cuadro 3.1: API de variables usadas en este proyecto.

Cabe señalar que este API es el mismo en las distintas fuentes anteriormente descritas. Las variables en negrita son las empleadas en el sistema descrito en esta memoria.

3.5. La biblioteca XForms

Las aplicaciones sobre robots suelen incluir normalmente una interfaz gráfica. A través de esta interfaz, se podrá visualizar los datos sensoriales o de los actuadores que el robot va transmitiendo. Además, el usuario podrá interactúar y depurar la aplicación con facilidad y rapidez. Cabe señalar que el comportamiento autónomo del robot no depende de esta visualización.

Uno de los sistemas gráficos más extendidos, y el mayor dentro de Linux es el sistema de ventanas `X-Windows`. Dentro de este sistema se encuentra la biblioteca `XLib`. Esta biblioteca permite máxima flexibilidad y control en todos los detalles de la interacción gráfica, aunque se considera compleja. Por este motivo, han surgido otros paquetes que se construyen por encima de `XLib` ofreciendo gráficos predefinidos y un repertorio acotado de patrones de interacción y de visualización. La biblioteca *XForms* es uno de estos paquetes, y es el empleado para realizar la interfaz gráfica de este proyecto, puesto que además de ser una biblioteca sencilla de usar y flexible, es la librería usada dentro de la plataforma *JDE* para crear los esquemas de visualización. Otras posibles alternativas hubieran sido `Qt` o `GTK+`, pero por la implicación anterior se descartaron.

De igual modo, *XForms* ofrece una herramienta visual, *fdesign*, para crear la interfaz de manera sencilla. Con solo arrastrar los objetos al panel estaría creada la interfaz, posibilitando también la configuración de cada uno de esos objetos (tamaño, forma, color, etc.) .

XForms ofrece un amplio repertorio de objetos gráficos con los que confeccionar la interfaz, cada uno con una serie de atributos y operaciones. Estos objetos no sólo pueden recibir eventos provenientes del usuario mediante el ratón o el teclado, sino que también pueden ser accesibles desde el código.

La biblioteca *Xforms* permite al programa de la aplicación muestrear de manera no bloquante el estado de la interfaz y mantener el control del flujo de ejecución. De esta manera la interfaz gráfica se inserta en los programas del robot como dos tareas: muestrear si el usuario ha producido algún evento, y refrescar la imagen que se está mostrando. Habrá que llegar a un compromiso entre el tiempo de muestreo de los botones y el de refresco. Un corto periodo de muestreo produce un sistema más reactivo y un frecuente refresco de la imagen implica una visualización vivaz. Pero se podrá llegar a consumir mucho tiempo y ralentizar al resto de las tareas, consecuencias no aceptables.

Por último, señalar que *jde.c* dispone de un esquema básico, `guixforms` (Figura 3.8). Este esquema es de visualización y consta de una interfaz gráfica implementada mediante esta biblioteca. En este sistema se ha hecho uso de este esquema, modificándose a nuestras necesidades de visualización dando paso al `guiscene`.

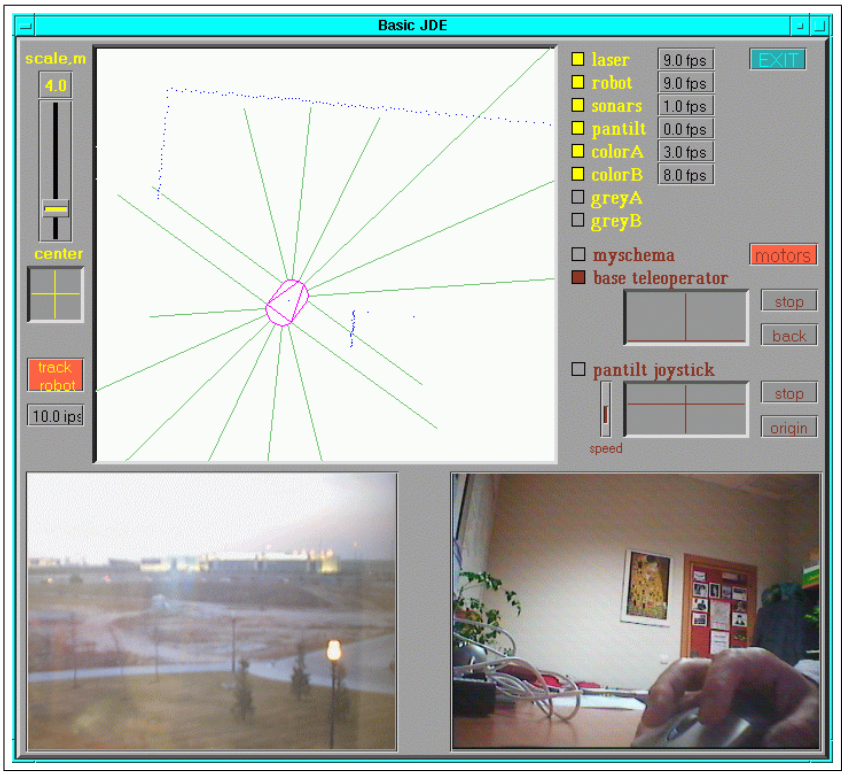


Figura 3.8: Esquema básico guixforms de jde.c

Capítulo 4

Descripción Informática

Tras haber explicado en capítulos anteriores los requisitos y las herramientas necesarias para la elaboración del proyecto, en el siguiente capítulo se detalla como se ha desarrollado el mismo. Para ello, se comenzará dando una visión del diseño global realizado y posteriormente se describirá el diseño y la implementación que se ha llevado a cabo en cada una de las partes.

4.1. Diseño Global con esquemas

Como se dijo en el capítulo 2, el objetivo de este proyecto es conseguir realizar un seguimiento de atención sobre diferentes objetos relevantes en una escena. Esto es, se debe conseguir captar nuevos objetos, alternar sobre los existentes y olvidarlos una vez que desaparezcan. También, se tiene que conseguir una representación de la escena del robot más amplia que el campo visual de una cámara, para ello se tiene la plataforma descrita en el capítulo 3, en la que hay una cámara sobre un cuello mecánico que va capturando imágenes cuando el cuello se posa en algún punto.

Como también se comentó en el capítulo 2, un requisito para realizar este proyecto es que se ha de seguir la estructura de la plataforma JDE[Cañas03]. En esta línea se ha resuelto objetivo como la creación de dos esquemas, el `guiscene` y el `scene`. Ambos esquemas se ejecutan concurrentemente y de manera iterativa. En el esquema `scene` cada iteración se produce con una frecuencia de `80 ms` y en el `guiscene` cada `100 ms`.

El esquema `scene` es un esquema perceptivo pero no pasivo, cuyo estímulo es contruir la imagen de escena del robot para realizar un control de atención sobre los

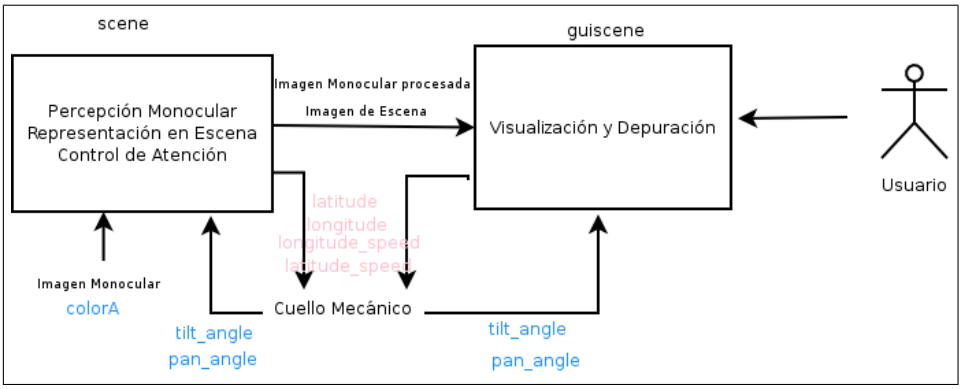


Figura 4.1: Division en Esquemas del sistema

objetos relevantes que se identifican en ella. Es un esquema perceptivo-activo porque necesita mover un actuador, en este caso el cuello mecánico, para percibir mejor su estímulo, construir la imagen en escena y la colección de objetos de interés en ella. Esta representación podrá ser usada por otros esquemas para materializarla en algún comportamiento, leyendo las variables necesarias que exporta `scene` para almacenar la imagen de escena.

Las entradas que recibe este esquema son las imágenes monoculares capturadas y la posición *(pan, tilt)* del cuello mecánico. Las imágenes monoculares son analizadas con técnicas de filtrado y segmentación para hallar objetos relevantes en ellas. Con estas imágenes procesadas se podrá constuir la imagen de escena. Ambas imágenes, la monocular y la de escena serán visualizadas en el esquema `guiscene`. En el `scene` además, se implementa el control de atención sobre los diferentes objetos. Los objetos identificados serán atendidos mediante comandos que se le enviarán al cuello mecánico. Esta funcionalidad está descrita en las secciones 4.2, 4.3 y 4.4.

El `guiscene` es un esquema de servicio. Es el encargado de visualizar una interfaz gráfica, desarrollada usando la biblioteca *XForms* (sección 3.5), que permite la activación y desactivación del anterior esquema. Por ello, una vez activado dicho esquema, ayuda en su visualización y depuración. Este esquema recibe como entrada la imagen monocular procesada y la imagen de escena para visualizarlas en la interfaz, ambas provenientes del esquema `scene`. También muestra las posiciones *pan, tilt* del cuello mecánico y permite su modificación. Toda esta funcionalidad se detalla en la sección 4.6

Para facilitar el desarrollo de estos dos esquemas, se han implementado dos

librerías escritas en *C*, una de ellas para la parte de análisis y tratamiento de imágenes *analisis_imagen* y la otra para ayudar en la representación de escena *representacion_escena*.

En las secciones siguientes se describe el diseño y la implementación de estos dos esquemas. En las cuatro primeras secciones se comentarán las tres partes en las que se divide el esquema *scene* y su implementación, y en la última la del esquema *guiscene*.

4.2. Percepción Monocular

Una de las partes en las que se divide `scene` es la percepción monocular, en donde se capturan y se analizan las imágenes con la finalidad de identificar los objetos relevantes que hay en la escena.

Para analizar las imágenes monoculares se han empleado cuatro etapas (Figura 4.3). Una primera es la **captura** o adquisición de las imágenes digitales por medio de una de las cámaras descritas en el capítulo 3. Para facilitar la captura de las imágenes, se ha empleado la arquitectura *JDE* descrita en el capítulo 3. La imagen se obtiene gracias a las variables que ofrece `jde.c`, en este caso `colorA`. Las imágenes se obtendrán a un ritmo superior a `30 imágenes por segundo` y con una resolución de `320x240`. El uso de `jde.c` ofrece que la recepción de las imágenes se produzca de manera independiente del hardware subyacente, es decir, con idependencia de si se trata de una cámara analógica con tarjeta capturadora o de una cámara firewire. Un ejemplo de una imagen recibida es la que se muestra en la figura 4.2.

En una segunda fase se ha pasado un **filtro** en el espacio *RGB* a la imagen, y se han filtrado los colores que se han considerado adecuados. En este proyecto se ha optado por los colores rosa y azul ya que son unos colores llamativos y fáciles de identificar, pero se puede cambiar a otros. En la tercera etapa se ha realizado la **segmentación**, esta es una basada en histogramas con doble umbral. La segmentación se usa para aislar los elementos que nos interesan de la imagen agrupando los píxeles de un determinado color. Por último, se han **identificado** los objetos segmentados hallando sus coordenadas, todo ello para realizar un futuro control de atención sobre ellos.

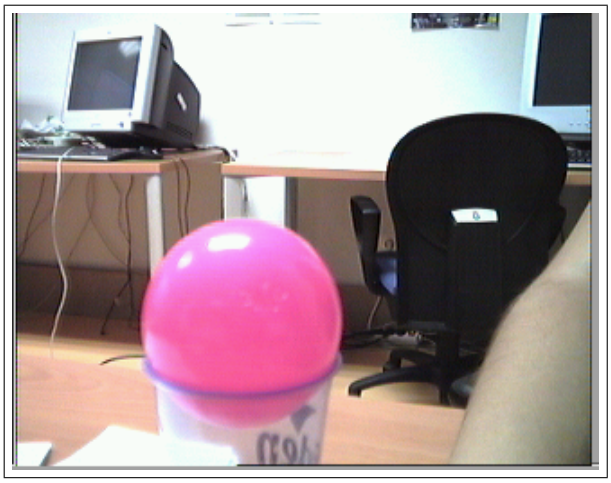


Figura 4.2: Imagen capturada por una cámara

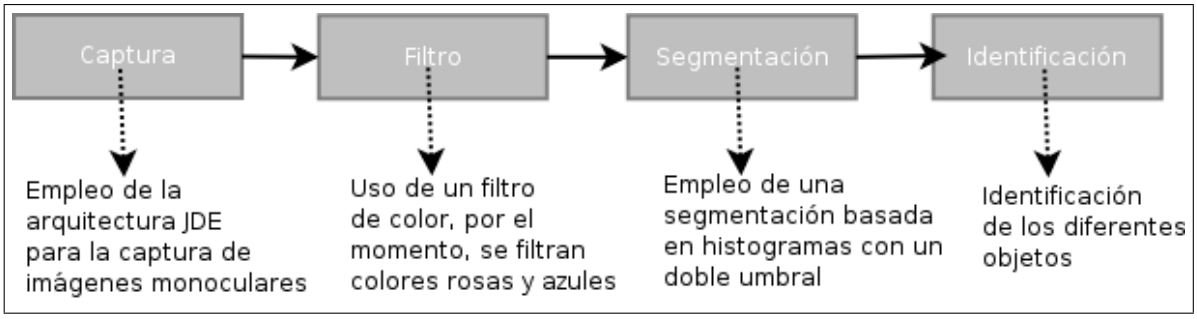


Figura 4.3: Etapas empleadas en el tratamiento de las imágenes monoculares

En cuanto a la implementación de estas cuatro fases se ha desarrollado una biblioteca escrita en *C*, *analisis_imagen*, que contiene funciones para el filtrado, la segmentación y la identificación de objetos. De esta forma podrá ser reutilizada en posteriores sistemas.

4.2.1. Filtro de Color

Como se ha explicado antes, a la hora de hallar los objetos relevantes en la escena, como estos son característicos por su color, se ha empleado un filtro de color en el espacio *RGB*. Se realiza este filtro para realzar las partes de la imagen que posteriormente vamos a analizar (Figura 4.4).

Las imágenes que se obtienen de la plataforma *JDE* están en un formato ***RGB***. Según este formato, cada píxel ocupa tres bytes correspondientes a la terna *RGB*: un byte es para la componente roja (**R**ed), otro para la verde (**G**reen) y un tercero para la azul (**B**lue). Aprovechando este formato y teniendo en cuenta que uno

Figura 4.4: Imagen monocular capturada por la cámara (Izquierda) y esa misma imagen filtrada (Derecha).

color	Rmax	Rmin	Gmax	Gmin	Bmax	Bmin
Rosa	255	200	140	0	255	100
Azul	100	50	200	100	255	150

Cuadro 4.1: Rango de valores RGB para los colores rosa y azul

de los requisitos primordiales de este proyecto es que el tratamiento de imágenes sea en tiempo real, se ha decido emplear un filtro de color en el espacio *RGB* a la imagen. Una posible alternativa sería el usar un filtro en el espacio ***HSI***, pero debido a que éste es más costoso que el anterior, ya que *JDE* nos ofrece estas imágenes en ese formato, y que el filtro de color en el espacio *RGB* se ha comportado bien frente a cambios en la luminosidad, se ha optado por seguir usando este filtro.

El filtrado de color consiste en ir comprobando píxel a píxel que las componentes R, G y B entran dentro de un cierto rango de valores. Este rango de valores para cada una de las componentes viene representado en la tabla 4.2.1. Estos valores se han ajustado experimentalmente dentro del escenario del laboratorio.

Una vez realizado el filtro a la imagen, y para facilitar la depuración del programa, se ha optado por representar los píxeles excluidos del anterior rango por su color original convertido a una escala de gris y los demás por un rosa o un azul dependiendo de su color original.

4.2.2. Segmentación

La segmentación es un proceso que consiste en dividir una imagen digital en regiones homogéneas con respecto a una o más características, en este caso el color, con el fin de facilitar su posterior análisis y reconocimiento. La técnica que se ha decidido emplear para segmentar, es una basada en histogramas. Un histograma es un diagrama de barras que contiene frecuencias relativas a alguna característica. En este caso por cada objeto que se quiera segmentar, se tienen dos histogramas (Figura 4.5), uno para las columnas (*histograma X*) y otro para las filas (*histograma Y*). En el *histograma X*, se tiene representado en el eje de abcisas, cada columna. Y en el eje de ordenadas, la frecuencia con la que aparece un píxel de un determinado color (rosa y azul) en una columna. El *histograma Y* es homólogo, con la salvedad que en el eje de abcisas estará representadas las filas en vez de las columnas. Se debe calcular ambos histogramas para cada color que se está buscando. El motivo por el que se hallan estos histogramas, es para enventanar las zonas con alta densidad de píxeles relevantes, para agrupar las zonas que son rosas y azules.

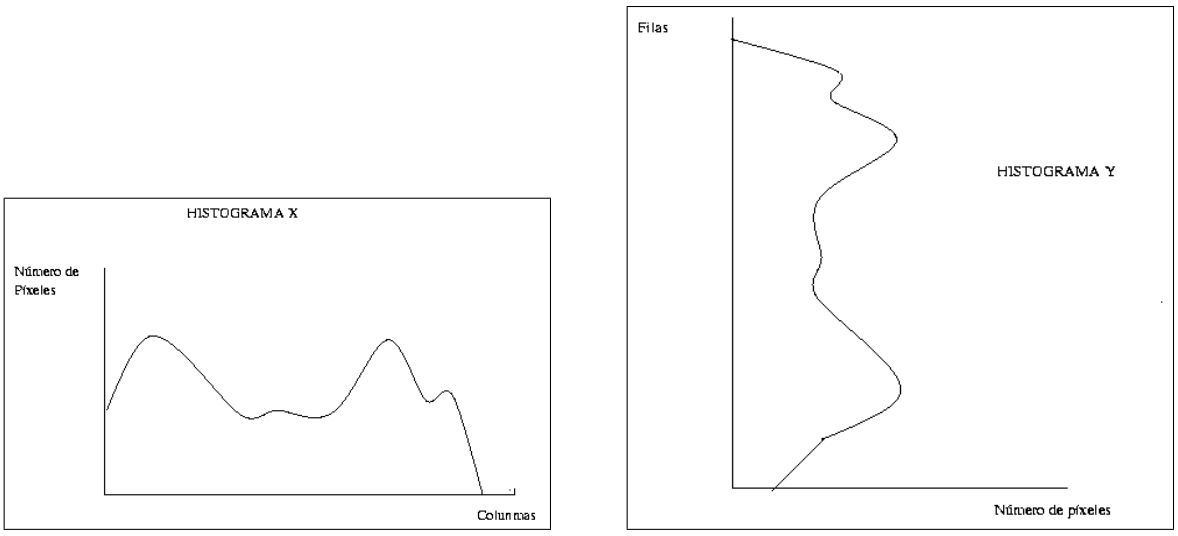


Figura 4.5: A la izquierda el histograma en el eje X y a la derecha el histograma en el eje Y.

Una posible técnica de segmentación desde el histograma consiste en utilizar un umbral simple, pero se descartó esta idea debido a que utilizar un umbral simple acarrea dos problemas. Por un lado el poner un umbral alto quitaría parte de los objetos, y por el otro, uno bajo cogería demasiado ruido (Figura 4.7). Por ello se decidió aplicar un doble umbral a los histogramas [SanMartin02][Martinez03] (Figura

4.6). Cabe señalar que la razón por la que se ha realizado este tratamiento independiente a cada eje, es para que el proceso de segmentación se efectúe de manera más rápida y eficiente.

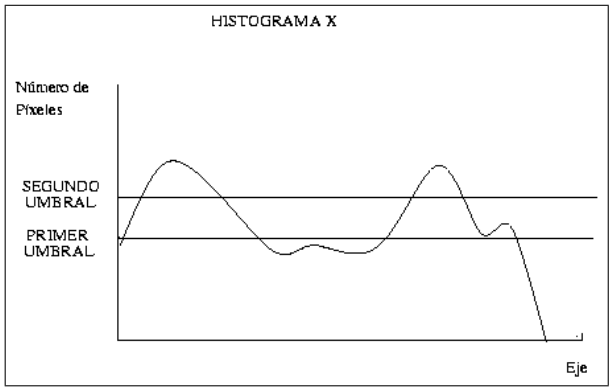


Figura 4.6: Histograma al que se le ha realizado un Doble Umbral

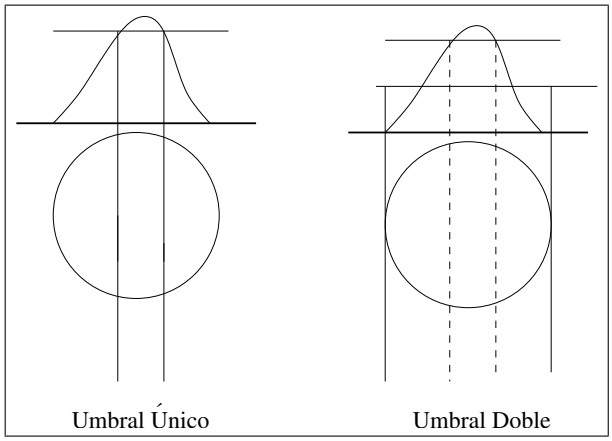


Figura 4.7: Comparativa entre usar un umbral único y uno doble

Para obtener los límites iniciales de cada zona, se apunta la columna o fila que supera el primer umbral, siendo el fin de la zona el punto en donde se pasa por debajo de ese umbral. Se va cogiendo las restantes zonas hasta el final del histograma. Esto dará varias zonas, pero sólo se guardarán aquellas zonas en las que en algún momento, además de pasar el primer umbral, pase también un segundo. En la figura 4.6 se puede ver como tres zonas pasan el primer umbral, pero sólo dos además pasan el segundo, por lo que sólo se guardarán esas dos. El segundo umbral asegura que hay un número relevante de píxeles del color que se anda buscando. Los valores de ambos umbrales se ve en la tabla 4.2.2

Umbral Mínimo	Umbral Máximo
10	20

Cuadro 4.2: Rango de valores RGB para los colores rosa y azul

Este algoritmo se pasa tanto en el histograma X como en el Y, para hallar los puntos que definen cada ventana, tal y como aparece en la figura 4.8. Como se puede ver en esa figura, se pueden obtener varias ventanas y alguna de ellas vacías (ventanas fantasmas), para solucionar este problema, se guardan aquellas ventanas que posean un número significativo de píxeles del color que se busca en su interior, en este caso ese número de píxeles es `80`. Se puede ver un ejemplo de una imagen resultante tras la segmentación en la figura 4.9, donde se han pintado los contornos de las ventanas, además de los píxeles de color.

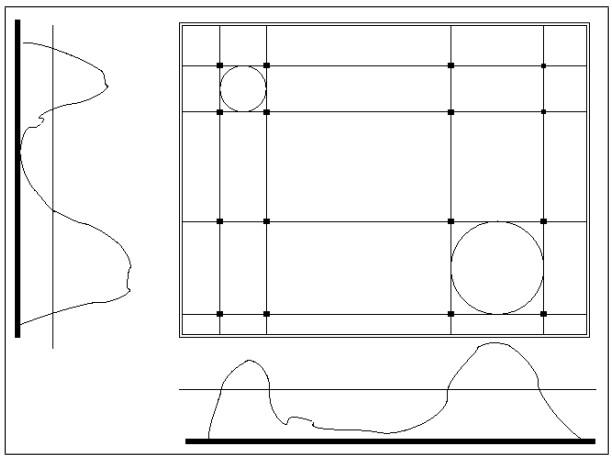


Figura 4.8: Ejemplo del eventanado

Para finalizar la parte de percepción, comentar como se han identificado los objetos partiendo de las imágenes segmentadas. Tras la segmentación, se han obtenido una serie de ventanas en la imagen, una por cada objeto identificado. El interés que se posee en esta parte es el obtener las coordenadas del objeto en la imagen para posteriormente realizar un control de atención sobre él.

Para hallar tales coordenadas habrá que hallar la coordenada central de la ventana. Para ello, sabiendo que cada ventana tiene cuatro vértices (Figura 4.10),

$$P_1(x_1, y_1), P_2(x_2, y_2), P_3(x_3, y_3) y P_4(x_4, y_4)$$

, se obtiene la coordenada central de la siguiente manera:

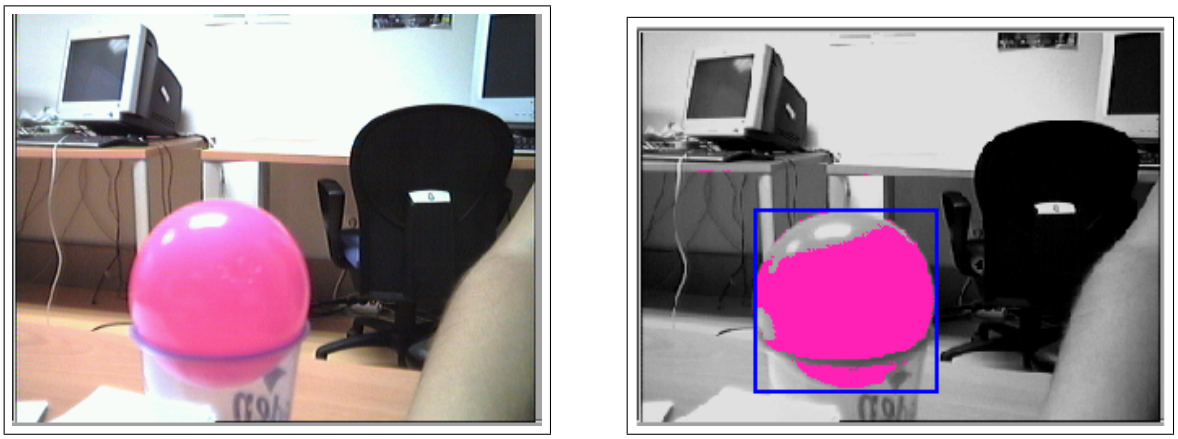


Figura 4.9: Imagen monocular obtenida por la cámara (izquierda) e imagen segmentada (derecha)

$$P_{centro} = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_3}{2} \right)$$

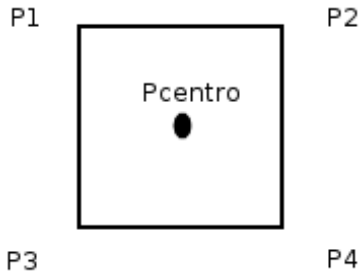


Figura 4.10: Ventana con sus cuatro vértices y el punto central.

4.3. Representación de la Escena

Otra de las partes del `scene` es el tener una representación visual de la escena. El alcance de la imagen de escena es mucho mayor que el campo visual de la cámara monocular (figura 4.11), por ello se ha de crear dicha imagen. Para ello, se tiene la plataforma descrita en el capítulo 3, un robot *Pioneer* sobre el que se sitúa un cuello mecánico y sobre éste, a su vez, hay una cámara que irá capturando imágenes. Gracias al cuello, se podrá ir moviendo la cámara en diferentes orientaciones para poder completar la imagen de escena.

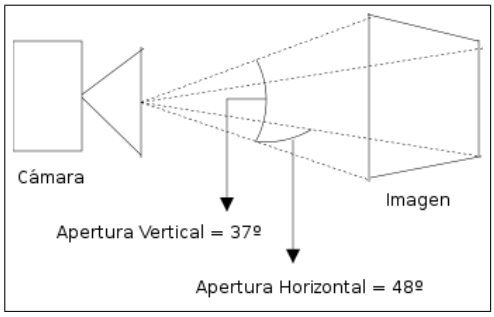


Figura 4.11: Apertura horizontal y vertical de la cámara empleada.

Este movimiento del cuello mecánico es en horizontal (***pan***) y en vertical (***tilt***), describiendo por tanto una especie de cúpula (Figura 4.12). Si por cada una de las posiciones que puede tomar el cuello (coordenadas *(pan, tilt)*) se tomara una imagen con la cámara de abordo, se tendría una gran imagen de escena compuesta por pequeñas imágenes monoculares.

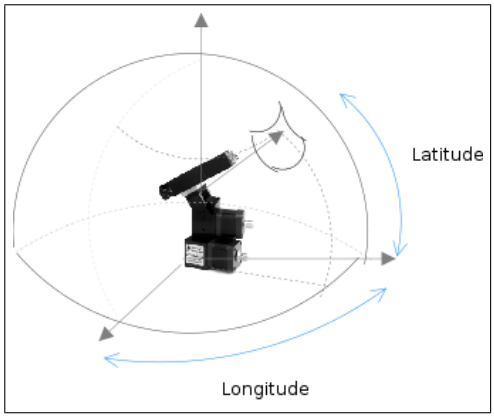


Figura 4.12: Cúpula descrita mediante el movimiento del cuello mecánico.

Ahora bien, para formar esta imagen de escena hay que realizar una correspondencia entre los píxeles de la imagen monocular con los píxeles de la imagen de escena, ya que la imagen monocular se proyecta en esta segunda. Para ello, hay que tener en cuenta que la imagen monocular, obtenida por la cámara, tiene coordenadas cartesianas *(u, v)* (figura 4.14), pero por otro lado, la imagen de escena tiene como coordenadas (α, β). Para poder construir esta imagen de escena, hay que transformar cada coordenada *(u, v)* de la imagen monocular (correspondientes a cada uno de sus píxeles) a las coordenadas (α, β) de la imagen de escena. Posteriormente (sección 4.6), para visualizar esta imagen de escena habrá que pasar estas coordenadas (α, β) a unas nuevas *(u', v')*, ya que la visualización seguirá un modelo cartesiano.

Figura 4.13: Imagen de Escena.

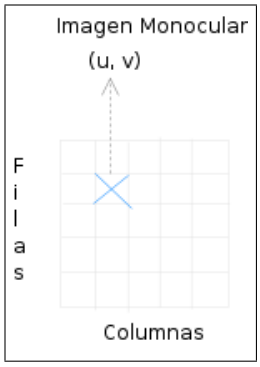


Figura 4.14: Coordenadas de la imagen monocular capturada.

Para convertir las coordenadas *(u, v)* de la imagen monocular a coordenadas (α, β) hay que observar los dos gráficos de la figura 4.15. Se sabe que en la columna 0 de la imagen monocular tiene como valor α: $\left(\frac{\Delta\alpha}{2}\right) + pan$ y el valor α en la última columna de la imagen tiene como valor: $pan - \left(\frac{\Delta\alpha}{2}\right)$, siendo *pan* el valor *pan* que tiene el cuello en ese momento y $\Delta\alpha$ el valor de la apertura de la cámara en horizontal, que en el caso de la cámara firewire (que es la empleada en la plataforma final) es de 48° (figura 4.11).

Con esto se obtienen dos puntos $P_1\left(0, \frac{\Delta\alpha}{2} + pan\right)$ y $P_2\left(ancho, pan - \frac{\Delta\alpha}{2}\right)$. Con estos dos puntos se obtiene la recta r. Los valores del eje de abcisas se corresponden con cada una de las columnas de la imagen monocular. Para calcular que valor α le corresponde a cada columna, se halla la ecuación de la recta r. Esta ecuación es:

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) \Longrightarrow \alpha = \frac{P_2.y - P_1.y}{P_2.x - P_1.x}(u - P_1.x) + P_1.y$$

En ella se sustituye el valor u por la columna y se haya el valor α.

Para hallar los valores β se hace básicamente lo mismo, como se puede ver en el gráfico derecho de la figura 4.15. Señalar que el valor de $\Delta\beta$ se correspondería con la apertura vertical de la cámara firewire, cuyo valor es 37° (figura 4.11).

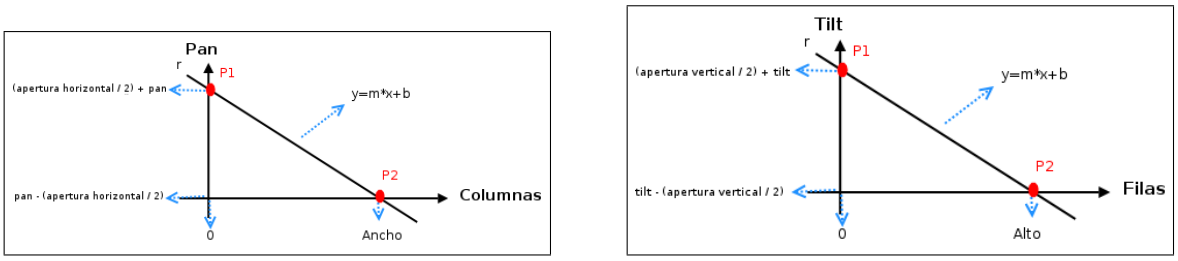


Figura 4.15: Ecuaciones de la recta para pasar de coordenadas (u, v) de la imagen monocular a coordenadas (α, β).

Las imágenes monoculares con las que se forma la imagen de escena están ya procesadas mediante las técnicas explicadas en la sección anterior (sección 4.2). Al proyectar las imágenes ya procesadas en la imagen de escena, la imagen de escena se almacenaría en un formato mixto, ya que los objetos relevantes aparecerán en color y lo que no es un objeto relevante en escala de grises. Además, esta imagen de escena se irá difuminando cada 6 iteraciones. Se difuminarán aquellas zonas que no han sido recientemente visitadas, de este modo la imagen de escena estará actualizada continuamente.

En la figura 4.16 se puede ver una visualización de una imagen de escena. Se puede apreciar como esta imagen está compuesta por imágenes monoculares procesadas. Para su composición el cuello mecánico realizó un barrido, de izquierda a derecha y de abajo a arriba tomando imágenes la cámara una vez que el cuello se posó en su punto destino.

En cuanto a la implementación, tanto de la composición de la imagen de escena, como de la conversión de coordenadas, ha sido creada una biblioteca escrita en *C*, *representacion_escena*, para tal fin. Contiene las funciones necesarias para la conversión de coordenadas y para la composición de la imagen de escena.

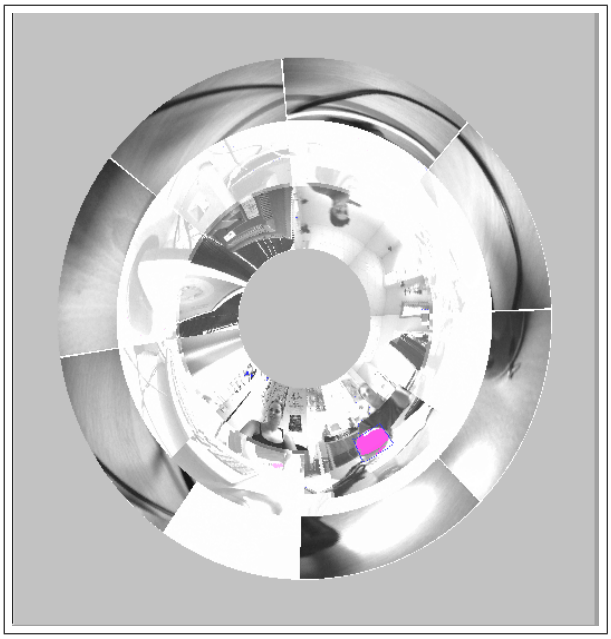


Figura 4.16: Imagen de Escena.

4.4. Control de Atención

Una de las partes más importantes de la que consta el presente proyecto es el control de atención. Para atender a varios objetos hay que hacerse varias preguntas: *¿Cuántos objetos se están atendiendo? ¿Dónde están situados? ¿A cuál de ellos hay que mirar a continuación?*.

Ahora bien, si sólo tuviéramos el campo visual de una cámara, habría objetos que no estarían en ese campo visual por lo que se olvidarían. Para ello se debe mover el cuello mecánico, para que en vez de sólo abarcar el campo visual monocular, poder abarcar toda la escena. El movimiento del cuello debe refrescar todos los objetos de la escena observándolos de vez en cuando, para realmente seguirlos. También debe buscar nuevos objetos que puedan aparecer en la escena. Pero, *¿cómo se gobierna ese movimiento del cuello mecánico?* empleando dos dinámicas concurrentes, la dinámica de saliencia y la dinámica de vida. También utilizando objetos y puntos de antención. La dinámica de saliencia, como se verá en la sección siguiente, permitirá alternar entre los distintos puntos de atención y saber a qué punto hay que dirigirse en todo momento. La dinámica de vida permitirá saber cuantos objetos hay en la escena y si ha desaparecido alguno.

En las subsecciones siguientes se describe en profundidad el diseño de las diferentes partes en las que se descompone el control de atención.

4.4.1. Dinámicas

Para gobernar el movimiento del cuello mecánico se han introducido dos dinámicas y objetos y puntos de atención. Los objetos representan a los objetos relevantes en la escena. Son relevantes porque poseen una característica que les hace especial. En este caso se ha elegido que esa característica sea el color, y en especial serán los colores rosa y azul.

En cuanto a los puntos de atención, estos representan aquellos focos a los que se debe dirigir el cuello. Pueden ser o bien los objetos relevantes en la escena, o bien focos de atención para explorar la escena.

Las dinámicas empleadas para el gobierno del movimiento del cuello son dos, la dinámica de saliencia y la de vida, ambas se explican a continuación.

Dinámica de Saliencia

Saliencia es todo aquello que llama la atención o que sobresale en una situación determinada, por ello el foco de atención podrá ir variando a lo largo del tiempo.

En este sistema la saliencia indicará aquello qué puntos de atención deben ser visitados en qué momento. Si tuviéramos un punto de atención con una saliencia muy alta será el visitado ya que es un punto un punto que llama la atención. En contra si ésta fuera baja, no sería visitado. Una forma de decidir la saliencia que posee cada punto de atención en función del tiempo que hace que no se visita. Un punto que hace tiempo que no se ha visitado causará mayor atención que uno que se ha atendido recientemente. Por ello si hay un punto que hace tiempo que no se ha visitado, debería tener mayor saliencia que los que acabamos de visitar.

La implementación de esta dinámica consiste en que cuando se visita un punto, su saliencia disminuya drásticamente y la de los demás puntos no atendidos se incrementarán. Ésto se ve representado en estas dos ecuaciones:

$$sal(punto, t) = sal(punto, t-1) + \Delta S_{time}$$

$$sal(punto, t) = 0$$

Con esta implementación, el punto que es atendido irá variando, puesto que cada vez habrá un punto diferente con mayor saliencia.

Dinámica de Vida

La otra dinámica es la vida. Con esta dinámica lo que se pretende saber es cuando un objeto ha salido de la escena y si aún sigue en ella. Para ello si la vida de un objeto es superior a un cierto umbral, es que todavía sigue en la escena, pero si está por debajo es que ha desaparecido.

La dinámica de vida es la que indica cuando hay que aumentar y cuando hay que decrementar la vida de un objeto. Su funcionamiento es inverso al de la saliencia. Un objeto frecuentemente visitado tendrá mayor vida que uno que apenas se visita. Si la vida de un objeto es inferior a un determinado umbral, este "morirá" y será, por lo tanto, olvidado y no se volverá a visitar.

Para implementar esta dinámica lo que se ha hecho es que cada vez que se visita un objeto su vida se incrementa (hasta un determinado umbral) y la de los objetos no observados en esa iteración disminuye. Esto se puede observar en las siguientes fórmulas:

$$vida(objeto, t) = vida(objeto, t-1) - \Delta V_{time}$$

$$vida(objeto, t) = vida(objeto, t-1) + \Delta V_{observacion}$$

4.4.2. Inserción de nuevos puntos de atención

En todo momento puede interesar la búsqueda de nuevos objetos relevantes en la escena. Para ello se deben insertar puntos de exploración. Esta búsqueda puede interesar por ejemplo al principio de la ejecución, momento en el que aún se desconocen las zonas de la escena en donde hay objetos a atender.

Estos nuevos puntos de exploración pueden ser de dos clases: ***Puntos de barrido*** y ***Puntos aleatorios.***

Los puntos de barrido irán siguiendo una trayectoria definida para garantizar un recorrido completo de la escena. La trayectoria elegida va desde la posición *pan* más

baja a la más alta y desde la posición *tilt* más baja a la más alta. De este modo si hay una concentración de objetos en una zona determinada, con esta trayectoria serán identificados más rápidamente.

Por otro lado, los puntos aletarios son generados aleatoriamente. Para ello, sus coordenadas *(pan, tilt)* deberán ser generadas aleatoriamente teniendo en cuenta que no debe haber dos puntos de atención iguales.

Ambos puntos de atención deberán tener una saliencia inicial alta para que sean visitados más rápidamente y de ese modo comprobar si en ellos existe algún objeto relevante. Si fuera el caso, estos puntos seguirían existiendo al asociarse estos puntos con los objetos encontrados.

Para saber en qué momento se ha de introducir uno de estos puntos se hace de la siguiente forma. En la interfaz, como se verá en la sección 4.6, se puede modular la probabilidad con la que se inserta un elemento aleatorio. Esta probabilidad de inserción va de 0 a 1. Si es 0 es poco probable que se inserte un nuevo punto y si es 1 es muy probable. Una vez modulada esta probabilidad de inserción en la interfaz, se creará un número aleatorio de 0 a 1 en cada iteración. De este modo si el número creado es mayor que la probabilidad de inserción, se insertará el punto y si es menor no se insertará.

4.4.3. Priorizar Objetos

El algoritmo diseñado permite asignar prioridades distintas a los diferentes colores relevantes, azul y rosa. Esto se ha materializado modificando la saliencia inicial que se le dan a los puntos de atención de los objetos. El color prioritario tendrá una mayor saliencia inicial que el otro color, lo que provocará que el cuello se pose más veces en aquellos objetos cuyo color sea el más prioritario. Esta prioridad se podrá modular mediante el interfaz gráfico de usuario (sección 4.6).

4.5. Implementación del esquema Scene

Tras haber descrito el diseño del esquema `scene`, en este capítulo se describe la implementación de este esquema. El estímulo de este esquema era el componer la imagen de escena y la colección de objetos identificados. Para ello se ha implementado

el algoritmo que se describe en esa sección. Con este algoritmo además de reconstruir la imagen de escena, se logrará un control de atención sobre la colección de objetos identificados.

En este algoritmo se han empleado las dos estructuras de datos comentadas en la sección 4.4, *objetos* y *puntos de atención*.

La información que se almacena de los objetos es su color en la escena (rosa o azul), su color de visualización, sus coordenadas *(pan, tilt)* y la vida que poseen.

En cuanto a la información que se requiere de los puntos de atención son: sus coordenadas *(pan, tilt)* y la saliencia que tienen.

Ambos son almacenados en dos listas, hay una lista para los objetos y otra para los puntos de atención. Se inserta un objeto en su correspondiente lista siempre que se identifique un objeto nuevo, y se elimina de la misma cuando su vida esté por debajo de un cierto umbral. La vida inicial que se le da a cada objeto es `1000` y la vida bajo la cual se considera que el objeto ha desaparecido es `995`. De igual forma, los puntos de atención se van insertando continuamente en su lista, o bien cuando se haya comprobado que en ese punto hay un objeto a atender, o bien se pueden insertar puntos de forma aleatoria, como se ha visto en la seccion 4.4.2. Todos estos puntos son eliminados de la lista cada vez que son visitados.

En cuánto a la descripción del algoritmo hay que observar el pseudocódigo 4.5 y la figura 4.17 que contiene el correspondiente diagrama de flujo.

Se tiene un bucle infinito en el que primeramente se comprueba si se inserta un nuevo punto de atención aleatorio. La manera en la que se inserta este nuevo punto es la explicada en la sección 4.4.2.

Tras ello, se obtienen las coordenadas del próximo destino del cuello mecánico. Para ello se obtienen las coordenadas del punto de atención con mayor saliencia, ya que es el foco que llama más la atención. Una vez visitado este punto es eliminado de la lista. Señalar que si las coordenadas de dicho punto superan una cierta distancia de

```
num_objetos=0;
num_puntos=0;
bucle
  insertar_punto_atencion ();    coordenadas=mayor_saliencia();
  si coordenadas ≠ coordenadas_actuales entonces
      mover_pantilt (coordenadas);
  fin si
  si coordenadas_pantilt = coordenadas entonces
      imagen_monocular = capturar_imagen ();
      filtrar_imagen (imagen_monocular);
      lista_obj_monocular=segmentar_imagen (imagen_monocular);
      identificar_objetos (lista_obj_monocular);
      incrementar_saliencia ();
      componer_imagen_escena (imagen_monocular, imagen_escena);
  fin si
fin bucle
```

Cuadro 4.3: Pseudocódigo del algoritmo para el control de atención

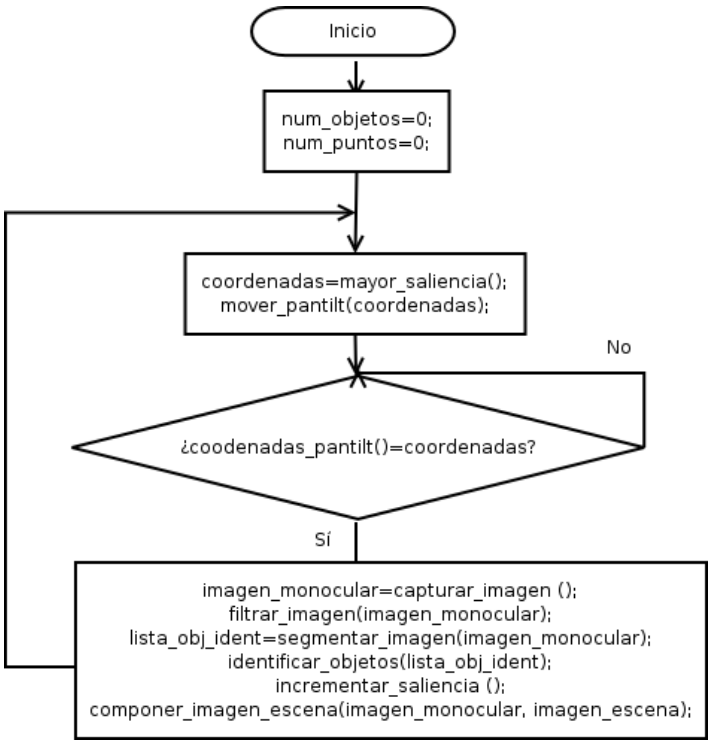


Figura 4.17: Diagrama de flujo del algoritmo

la posición en la que se encuentra el cuello en ese momento, el cuello se moverá a la nueva posición, sino se quedará donde está.

Con las anteriores coordenadas se le ordena al cuello mecánico que se mueva a esa posición. Cuando el cuello ha llegado a esa posición y está ya parado, se cap-

```
comprobar_objetos_olvidados ();
si lista_obj_monocular ≠ 0 entonces
para i=0 hasta i≥longitud(lista_obj_monocular) hacer
  si encaja (lista_obj_monocular[i], lista_objetos_escena) entonces
      actualizo_datos (lista_obj_monocular[i], lista_objetos_escena);
      insertar_nuevo_punto_atención (lista_obj_monocular[i].coordenadas, lista_puntos);
      incrementar_vida (lista_objetos_escena[posicion(lista_obj_monocular[i])].vida);
  sino
      insertar_nuevo_objeto (lista_obj_monocular[i], lista_objetos_escena);
      insertar_nuevo_punto_atención (lista_obj_monocular [i], lista_puntos);
      incrementar_vida (lista_objetos_escena[posicion(lista_obj_monocular[i])].vida);
  fin si
  decrementar_vida ()
fin para
```

Cuadro 4.4: Pseudocódigo para la función identificar_objetos

tura una imagen con la cámara de abordo. Esta imagen es tratada con las técnicas anteriormente descritas (sección 4.2) para identificar objetos relevantes. Estos objetos objetos son insertados en una lista auxiliar con el fin de ser analizados en la función `identificar_objetos()`.

Para explicar como se añaden los objetos identificados a la lista de objetos, y como se añaden nuevos puntos de atención, hay que observar el siguiente pseudocódigo (tabla 4.5) de la función `identificar_objetos ()` y su correspondiente diagrama de flujo en la figura 4.18.

Primeramente, en la función `identificar_objetos ()` se comprueba si hay algún objeto cuya vida haya disminuido por debajo de un umbral, en la función `comprobar_objetos_olvidados`. Si esto hubiera ocurrido, este objeto habría sido eliminado de la lista ya que habría que olvidarlo.

A continuación se comprueba si los objetos percibidos en la imagen monocular encaja con alguno que tengamos ya almacenados en la lista de objetos. Esto se hace comprobando si las coordenadas de los objetos percibidos guardan cierta distancia mínima con un objeto ya almacenado en la lista de objetos. Si es así, este objeto percibido encaja con el objeto almacenado y se sustituyen las coordenadas del objeto

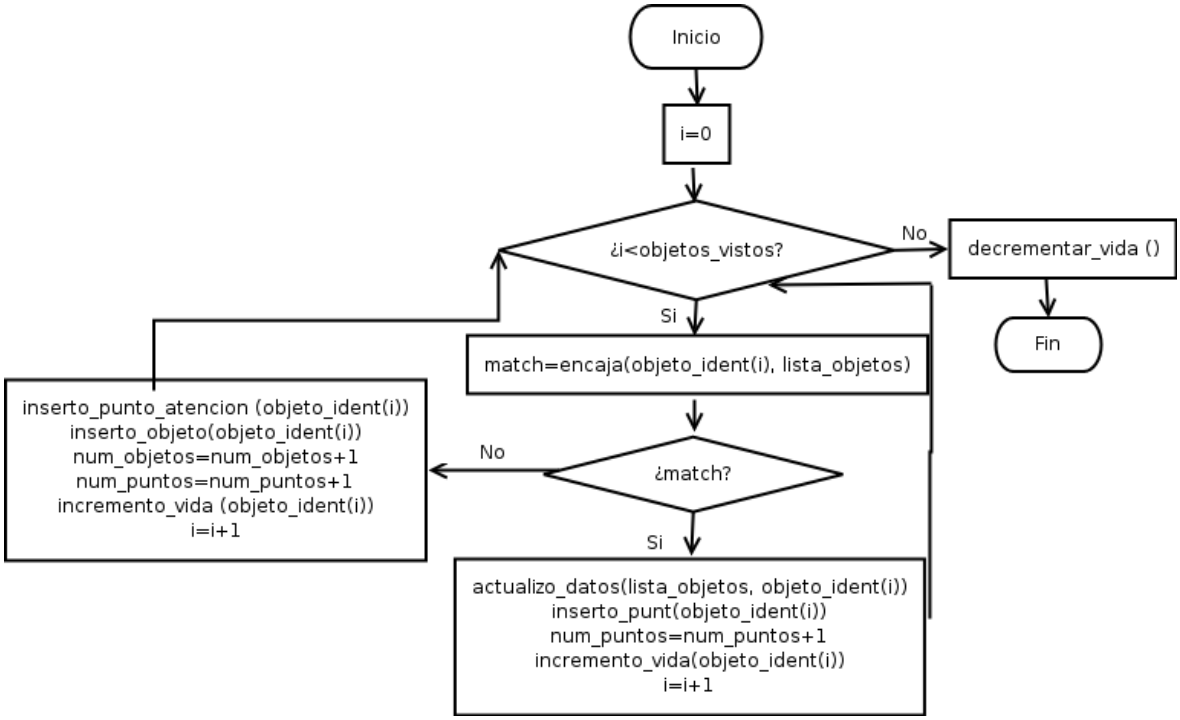


Figura 4.18: Diagrama de flujo de la función identificar_objetos ()

almacenado por las de nuevo objeto percibido. De este modo, si el objeto anteriormente identificado se movió, al ser percibido ahora y al encajar ambos, se podrán seguir objetos aunque se muevan en la escena. También, su vida se incrementaría al haber sido visitado. Sino encaja se añade directamente el objeto percibido a la lista de objetos como un nuevo objeto identificado, con sus correspondientes coordenadas. En ambos casos, se inserta un nuevo punto de atención a la lista con las coordenadas del objeto identificado y su saliencia inicial depende de la prioridad que tenga en ese momento el color del objeto (sección 4.4.3).

Tras comprobar objeto a objeto si encaja o no, se decrementa en 1 la vida de los objetos no percibidos en esa imagen.

Para finalizar el algoritmo, hay que retomar a la tabla 4.5. Una vez identificado los objetos hay que aumentar la saliencia en 1 de todos los puntos de atención. Tras ello, se inserta finalmente la imagen monocular capturada a la imagen de escena.

Básicamente este ha sido el algoritmo desarrollado para implementar el estímulo del esquema `scene`, construir la imagen de escena y tener la colección de objetos relevantes en la misma.

4.6. Visualización Gráfica con Guiscene

Como se dijo en la sección 3.5, las aplicaciones sobre robots suelen incluir una interfaz gráfica para permitir al usuario interactuar con el sistema, visualizar datos provenientes del robot y fundamentalmente, para facilitar el proceso de depuración y prueba. La interfaz de usuario de este sistema ha sido implementada mediante un esquema de visualización típico de la plataforma *JDE*, ***guiscene***, y se ha usado la biblioteca *Xforms* descrita en la sección 3.5. Esta interfaz se puede observar en la figura 4.19 y la funcionalidad que la misma ofrece se puede ver en el diagrama de casos de usos dibujado en la figura 4.20.

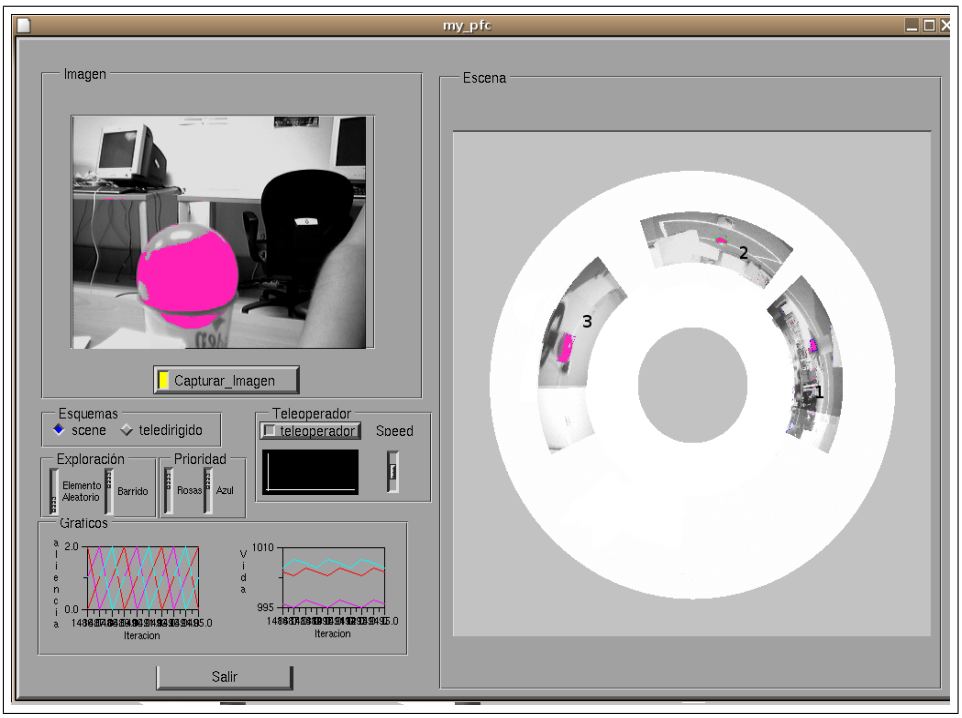


Figura 4.19: Interfaz Gráfica de Usuario del sistema

Esta funcionalidad es la siguiente:

- **Visualización de las imágenes monoculares procesadas:** Se visualizarán las imágenes recibidas en una ventana del propio interfaz. Esta imagen se mostrará ya filtrada y segmentada. El elemento identificado se mostrará en color, justo con el segmento de la segmentación, y el resto en escala de grises.

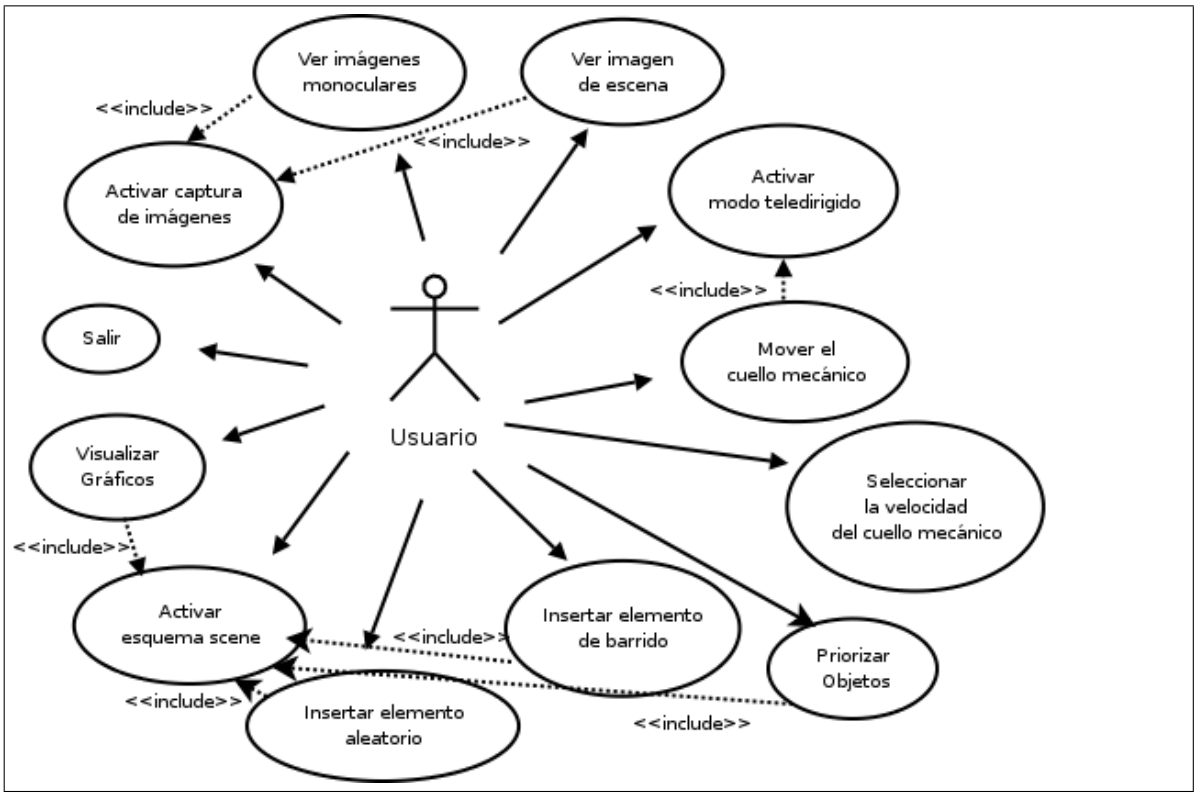


Figura 4.20: Diagrama de Casos de Uso en donde se expresa la funcionalidad de la interfaz.

- **Visualización de la imagen de escena:** Se visualizará la imagen de escena creada a partir de las anteriores imágenes monoculares procesadas. En la sección 4.3 se explicó como se proyectaba estas imágenes en la imagen de escena. La imagen de escena tiene como coordenadas (α, β) y a la hora de visualizarla, se hará con coordenadas *(u, v)*. Para realizar esta nueva conversión hay que observar la figura 4.21. Se quiere hallar el valor de *(u, v)*, que se corresponden con el valor de las coordenadas del punto B. Para hallarlas podemos ver que se forma un triángulo rectángulo con el eje de coordenadas que parte de la coordenada central de la imagen de escena. Gracias a ello se pueden usar las siguientes reglas trigonométricas:

$$\operatorname{sen}\alpha = \frac{AB}{BC} \quad \cos\alpha = \frac{AC}{BC}$$

La longitud del segmento AB se correspondería con el valor u y la longitud del segmento AC se correspondería con el valor de v. Para hallar la longitud del segmento BC, se tiene que hallar la longitud que tiene el radio de la circunferencia en ese punto, para ello se pasará el valor *tilt* que tiene el cuello, a este radio. Para ello se usará la ecuación de la recta representada en el gráfico de la izquierda de

la figura 4.21. El valor del ángulo α se corresponde con el valor *pan* del cuello en esa posición. Una vez obtenido el radio y el ángulo sólo habría que sustituir en las correspondientes fórmulas.

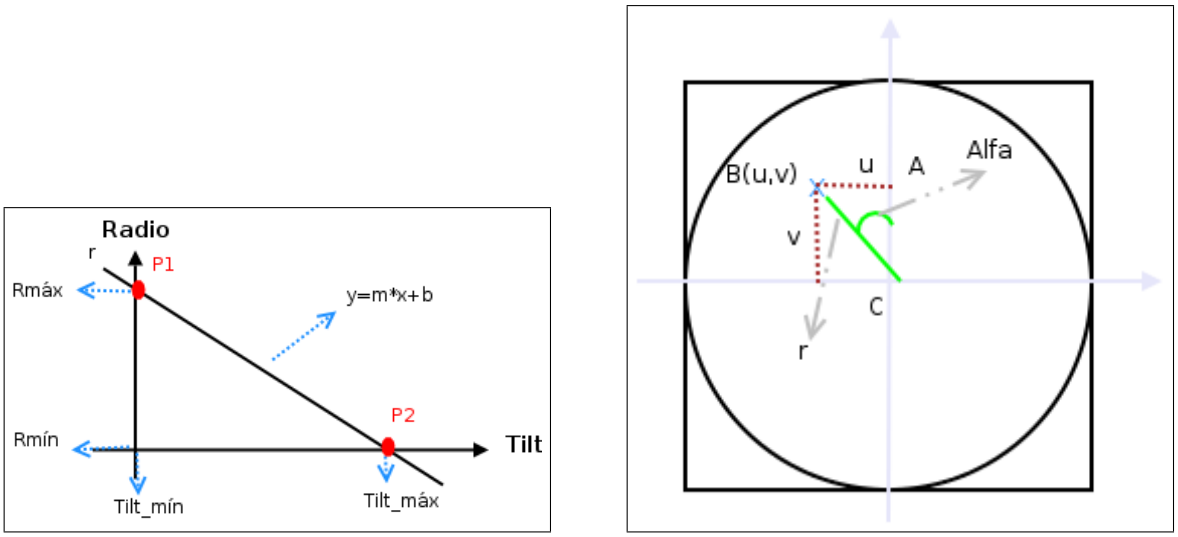


Figura 4.21: Transformación de las coordenadas (pan, tilt) a coordenadas (u, v) en la imagen de escena.

En la figura 4.16, que viene a continuación, se puede observar esa transformación.

- **Visualización de las dinámicas:** Se visualizará la vida y la saliencia de cada objeto mediante unos gráficos. Cada una de las funciones de vida y saliencia de cada objeto estará representada por el color de ese objeto.
- **Modular el esquema scene:**
  - *Fijar la velocidad del cuello:* Se podrá seleccionar la velocidad a la que queremos que se mueva el cuello mediante un slider.
  - *Cambiar la probabilidad de búsqueda aleatoria:* Se podrán asignar la probabilidad con la que un punto de atención aleatorio se introduce en la dinámica.
  - *Cambiar la probabilidad de búsqueda de barrido:* Se podrá, del mismo modo, asignar la probabilidad con la que un punto de de barrido se introduce en la dinámica.
  - *Modular la prioridad de los colores relevantes:* Se podrá elegir la prioridad, que tienen objetos de determinadas características, de ser atendidos.

  - *Activar el esquema scene:* Se podrá activar el esquema para el control de atención sobre los diferentes objetos de la escena.

- **Activar la Captura de Imágenes:** Se podrá activar la captura de imágenes mediante un botón del propio interfaz. Una vez activado, e recibirán imágenes moculares a través de las variables compartidas ya descritas en el capítulo 3.

- **Activar el modo teledirigido, para poder mover el cuello mecánico:** El usuario podrá activar el movimiento teledirigido del cuello mecánico, esto es, mover el cuello mecánico usando la interfaz

  - *Mover el cuello mecánico:* Se podrá mandar la posición a donde queremos que se mueva el cuello mediante dos formas diferentes. Una de ellas es con el joystick y la otra es haciendo click en la ventana en donde se muestra la imagen de escena.

- **Salir del Programa:** Se podrá parar la ejecución del sistema mediante el botón salir.

Capítulo 5

Pruebas

Tras haber descrito cómo se ha diseñado e implementado el algoritmo de atención, en este capítulo se valida y demuestra que el sistema funciona y ha cumplido su objetivo principal: realizar un control de atención sobre objetos relevantes en una escena y mantener actualizada la representación de escena. Para ello se han realizado una serie de pruebas, con un sólo objeto, con dos o más del mismo color y con varios de diferentes colores. Los videos de dichas pruebas están disponibles en la web[1].

Señalar la configuración empleada para los valores de los parámetros empleados en las dinámicas (sección 4.4.1) ha sido la siguiente: para la dinámica de vida, el incremento que se le da a la vida de un objeto depende del número de objeto que halla en la escena, ya que un incremento bajo implicaría perder objetos. Este incremento oscila entre 1 y 5 para mantener en vida de uno a cinco objetos. Por otro lado, el umbral bajo el cuál se considera que ha desaparecido un objeto es 995 y la cota de vida para llegar a saturación es de 1010. En cuanto a la dinámica de saliencia, la saliencia inicial que se le da a un punto de atención normal es 0, y su incremento es 1. La saliencia inicial que se le da a un punto de exploración es de 5. Con respecto a los objetos prioritarios, su saliencia inicial irá en función de la prioridad modulada en la interfaz. Todos estos parámetros han sido ajustado manualmente en base a los experimentos y se ha afinado su elección para que las dinámicas generaran las funcionalidades requeridas.

5.1. Experimentos con un objeto

En la primera prueba se tiene un sólo objeto en la escena y se va a comprobar como con el algoritmo y las dinámicas explicadas en el capítulo 4, el sistema es capaz de

[1] *http://gsyc.escet.urjc.es/robotica/pfc/pfc-atencionescena.html*

buscar dicho objeto explorando la escena, realizar un seguimiento sobre él y olvidarlo cuando este desaparezca.

Para la búsqueda del objeto se ha realizado una exploración de la escena. Para ello la dinámica descrita en el capítulo 4 insertaba puntos de exploración (sección 4.4.2). En este caso se emplearán puntos de barrido. Esto se puede observar en la figura 5.1. El cuello se va moviendo siguiendo una trayectoria, una vez que identifica un objeto, en este caso uno de color rosa, se puede ver como a la vez que lo atiende (poses 4, 6, 8 en la figura 5.2) sigue insertando más puntos de exploración con el objetivo de buscar nuevos objetos (poses 1, 2, 3, 5, 7 en la figura 5.2). A su vez, en la visualización de la imagen de escena se puede observar como en las zonas en donde no hay objetos relevantes se va difuminando y la zona en donde se ve el objeto perdura (figura 5.2), ya que está siendo refrescada continuamente al ser frecuentemente visitada.

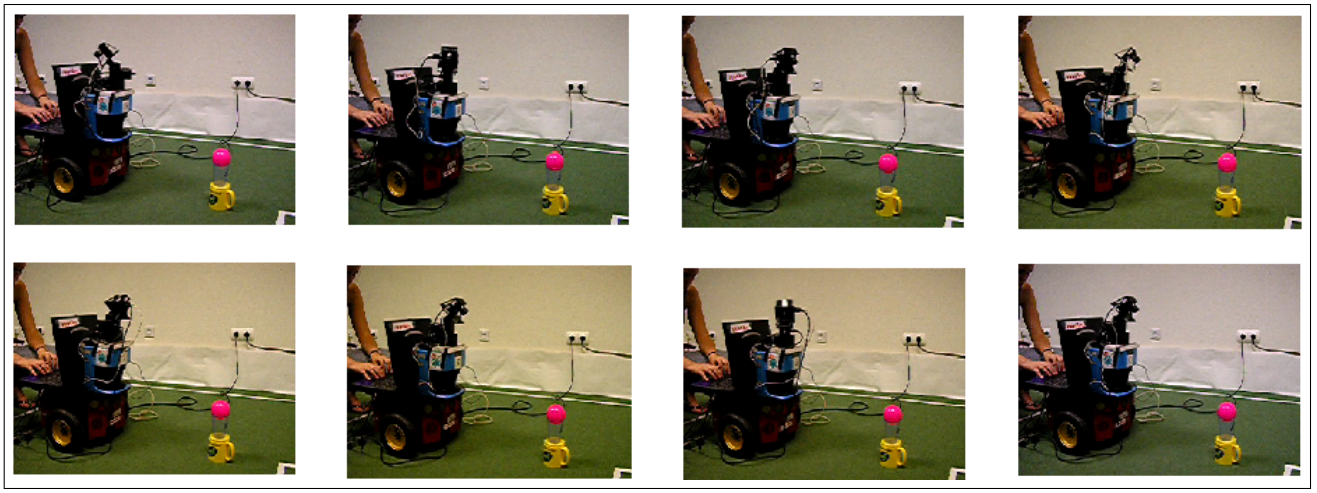


Figura 5.1: Barrido del cuello en busca de nuevos objetos

Una vez encontrado este objeto se decidió probar si el sistema era capar de seguirlo a la vez que seguía realizando la exploración. Para ello se movió el objeto. Este seguimiento se ha resuelto usando las dos dinámicas de vida y saliencia (sección 4.4.1), se podría haber resuelto usando la técnica clásica de seguimiento reactivo [Martinez03] [SanMartin02], pero el uso de las dos dinámicas imprime mayor potencia y permite el seguimiento no sólo el seguimiento de 1 objeto sino también el de 2, 3 o más sin retocar nada. Además genera funcionalidad nueva como el olvido, la alternancia entre los focos de atención, funcionalidades que no están en los algoritmos clásicos de seguimiento. En la figura 5.3 se puede observar este seguimiento.

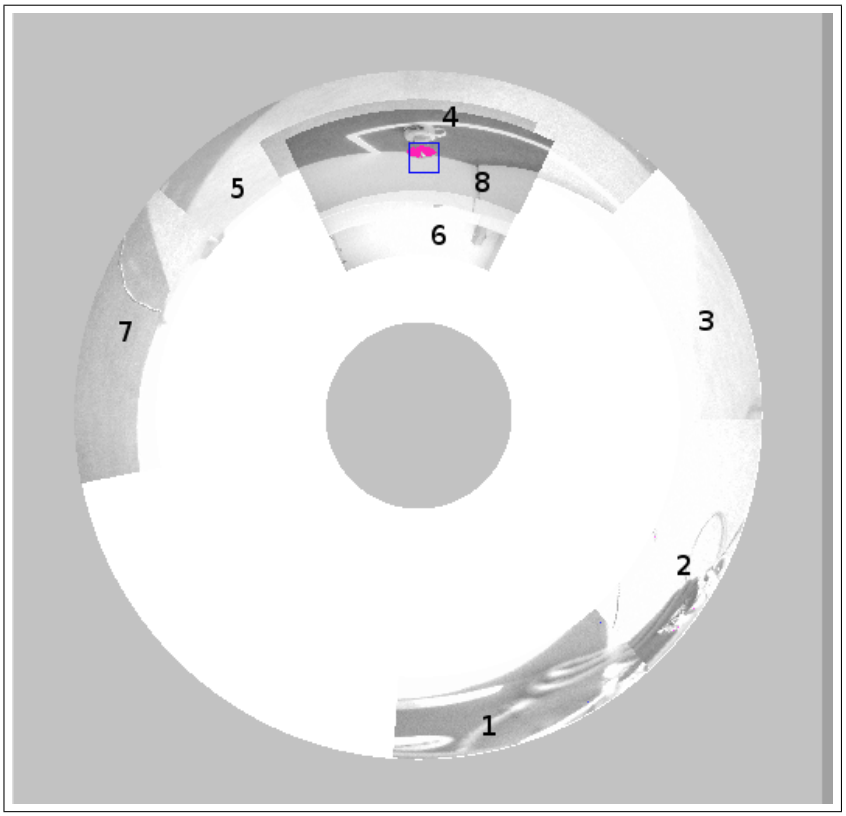


Figura 5.2: Visualización de la imagen de escena.

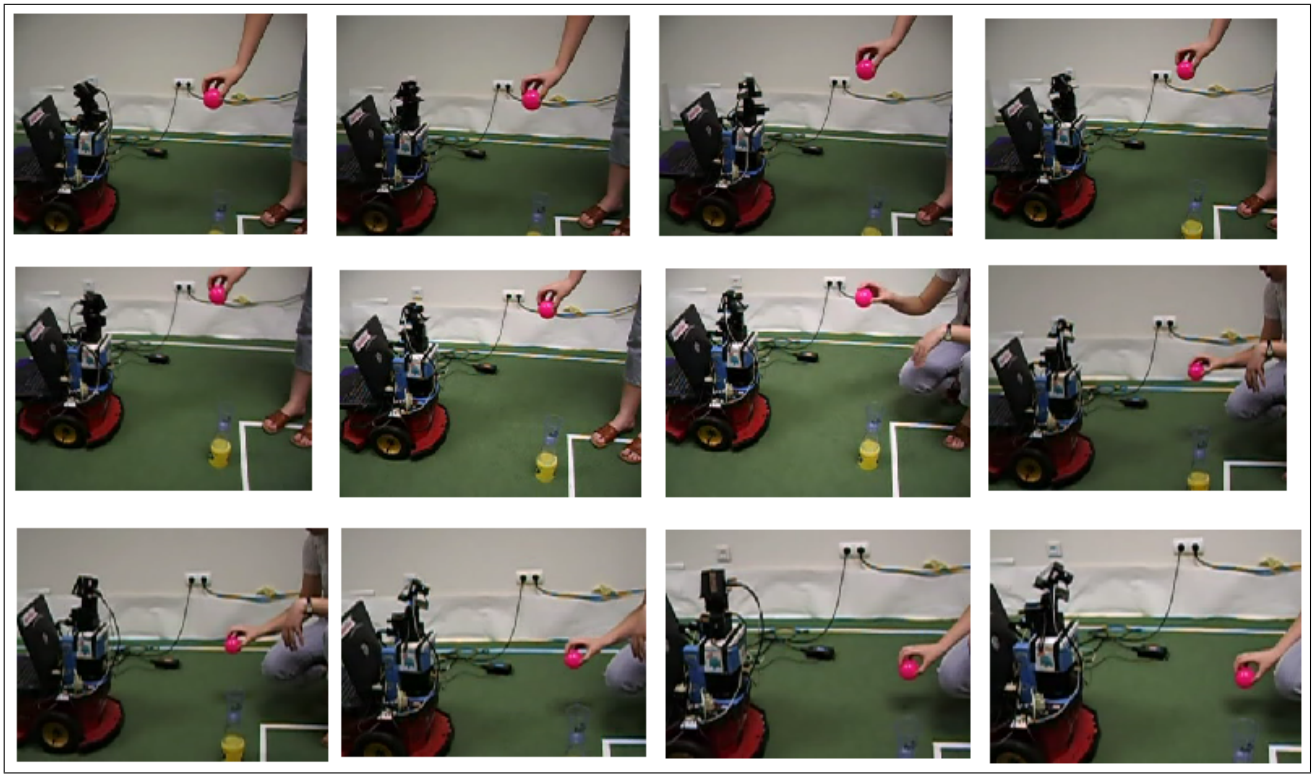


Figura 5.3: Seguimiento de un sólo objeto

Importate observar las gráficas de la saliencia (Figura 5.4) y la vida del objeto

(Figura 5.5). En la gráfica de saliencia (Figura 5.4) se puede observar como ésta, aunque tenga valores bajos, consigue un seguimiento correcto del objeto. Esto es así porque no siempre se explorará la escena (si mediante la interfaz se baja la probabilidad de exploración a 0) y será por lo tanto el único objeto. En esta ocasión la saliencia del objeto siempre es cero. Es cero porque al haber un único punto de atención que es el propio objeto, al ser observado la saliencia disminuye a cero, y como siempre es observado, esta saliencia siempre valdrá cero.

En la gráfica izquierda de la figura 5.5 se puede ver como inicialmente la vida del objeto va subiendo gradualmente hasta una determinada cota. Los picos que se observan en la figura son debido a que al haber puntos de exploración, cuando el cuello se posa en estos puntos, la vida del objeto decrece, y cuando se posa en el objeto aumenta. Por otro lado, cuando la pelota no se observa, la vida iría decrementándose (figura derecha 5.5) hasta un umbral mínimo bajo el que se olvidaría el objeto.

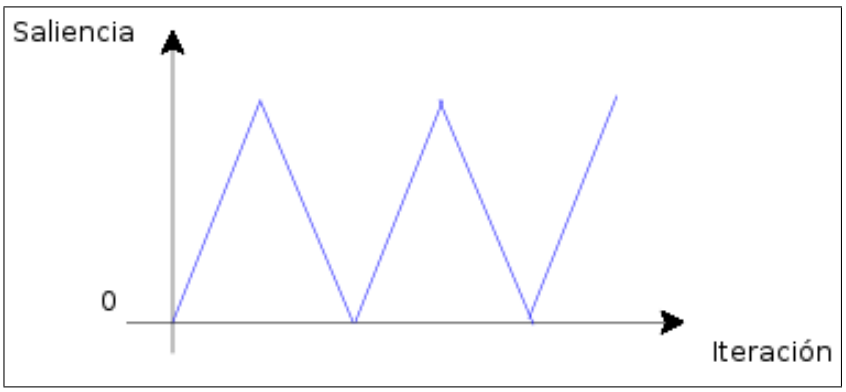


Figura 5.4: Saliencia de un sólo objeto

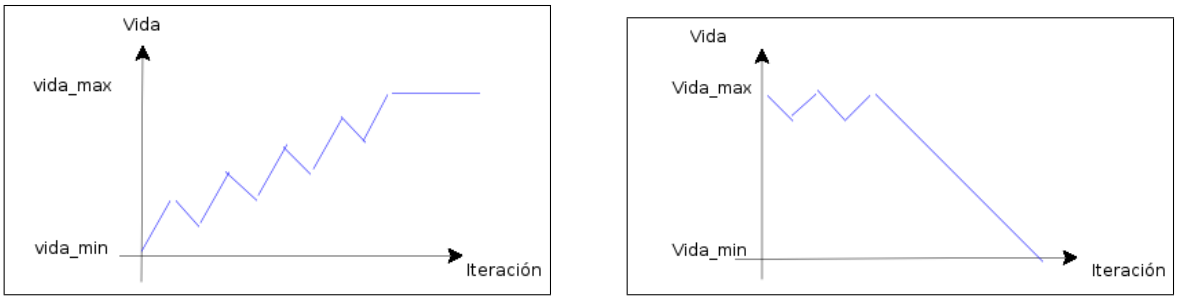


Figura 5.5: Vida de un sólo objeto

Destacar que a la hora de alcanzar el nivel de saturación en vida, no es igual

cuando se le proporciona un incremento de vida de uno que otro superior, cuanto más alto sea este incremento, antes se alcanzará el nivel de saturación (Figura 5.6).

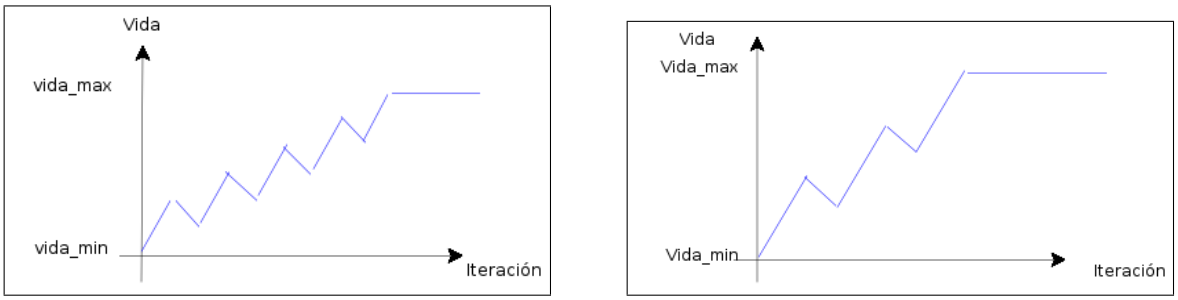


Figura 5.6: Incremento en vida pequeño (Izquierda) e incremento en vida grande (Derecha)

5.2. Experimentos con varios objetos del mismo color

En esta prueba se pretende comprobar como el sistema es capaz de realizar un control de atención sobre varios objetos, todos ellos del mismo color, en una escena. Para ello, al igual que en la prueba anterior, se va comprobar como se logra una búsqueda de objetos, se realiza una alternancia de seguimiento sobre ellos y es capaz de olvidar los objetos que desaparecen de la escena.

Para ello, se comienza igual que en la prueba anterior. Se han insertado puntos de exploración en los que se pretende buscar objetos a identificar. Una vez encontrados los objetos se decidió anular las probabilidades de exploración. Aunque el sistema funcione bien insertando dichos puntos, como se ha visto en la sección anterior, se ha simplificado para que las gráficas sean más nítidas y se ilustre mejor los aspectos de alternancia de seguimiento y olvido. Una vez eliminados estos puntos, un ejemplo de una imagen de escena resulta sería la de la figura 5.7, en la que se ve como se han identificado tres objetos en la escena.

Tanto si el número de objetos es dos, tres, cuatro o cinco se puede observar como el cuello va alternando entre cada uno de los objetos (Figura 5.8). Esto se ha conseguido gracias a la saliencia. En cada iteración la saliencia de los puntos de atención

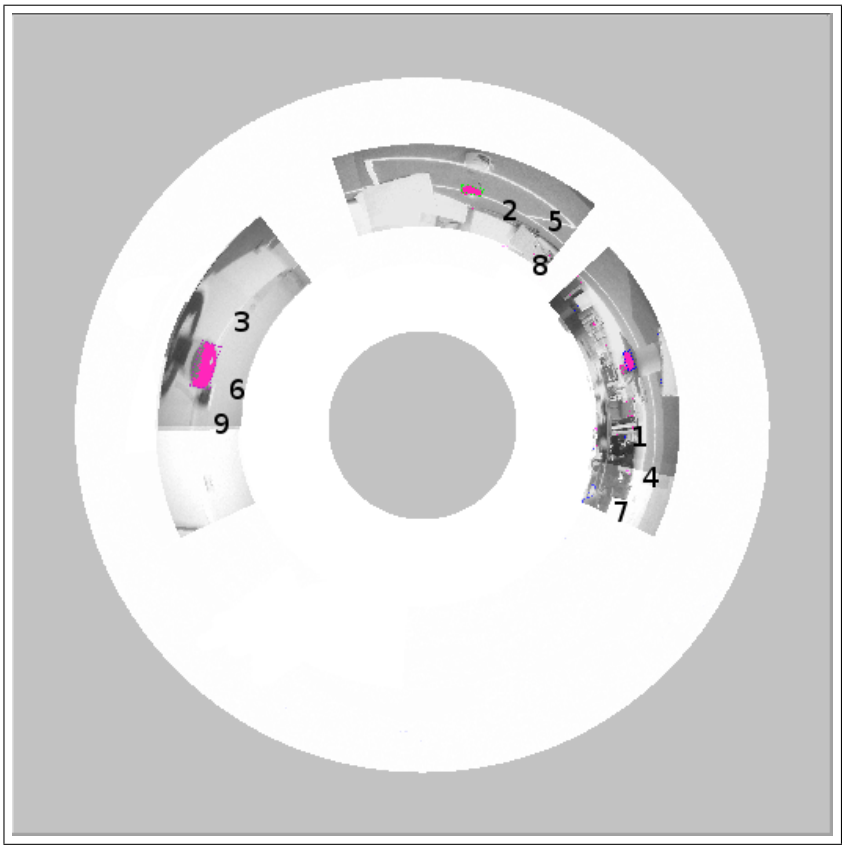


Figura 5.7: Imagen de Escena para tres objetos identificados

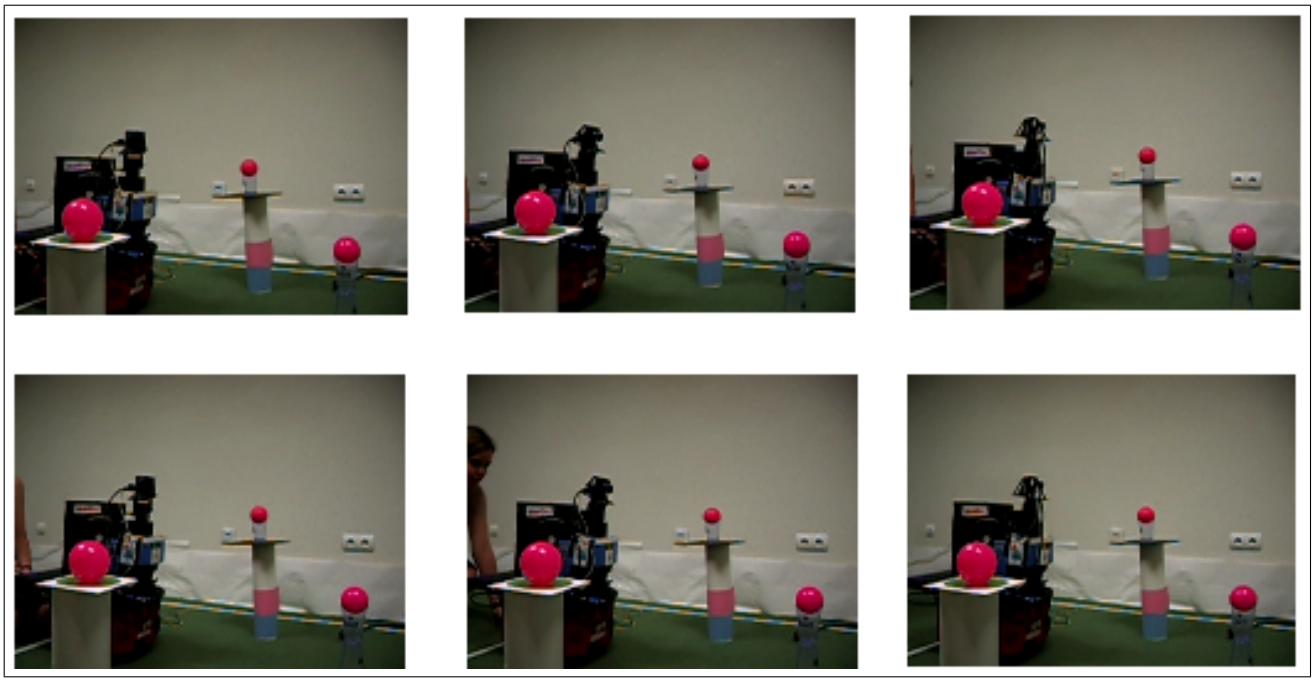


Figura 5.8: Seguimiento de un 3 objetos

correspondientes a los objetos no visitados se incrementa y la de los visitados disminuye a cero. Por lo que en cada iteración será un objeto diferente el atendido. Una vez que se

han atendido todos, se puede ver como se vuelve a atender a cada uno de ellos otra vez, siempre siguiendo el mismo orden. Esto se puede observar en la figura 5.9. Además, se puede ver como la gráfica de saliencia de dos puntos y la de tres es similar, en cada iteración hay un punto con mayor saliencia y el orden siempre es el mismo. Lo mismo ocurriría en el caso de cuatro y cinco puntos. Señalar que si algún objeto apareciera en la misma imagen que otro, sus correspondientes puntos de atención tendrían la misma curva de saliencia (figura 5.10), porque al observarse en la misma imagen en las mismas iteraciones, ambos incrementan y decrementan su valor de saliencia a la vez.

Figura 5.9: Gráfica de la saliencia con 2 puntos (izquierda) y con 3 puntos (derecha).

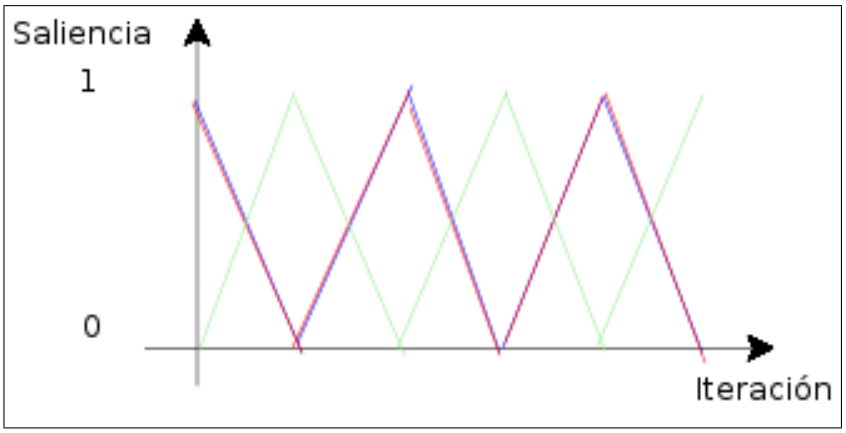


Figura 5.10: Saliencia de tres puntos en los que dos coinciden en la misma imagen.

En cuanto a la vida de los objetos, no es lo mismo el incremento de vida que se le debe de dar a dos objetos que el incremento con tres o cuatro objetos. Si este incremento es muy pequeño, en un sistema de tres o cuatro objetos se olvidarán objetos, ya que al no ser visitados en varias iteraciones seguidas, su vida disminuye de manera más rápida (figura 5.11). Con lo cuál necesitan de un incremento mayor para poder seguir siendo atendidos. Por ello la dinámica tiene un límite, en este caso cinco,

en cuanto al número de objetos atendidos. Para poder ampliar este número habría que modificar el valor del incremento de vida.

Por esa misma razón, al realizar la exploración de la escena, la frecuencia con la que busca objetos tiene que ser inversamente proporcional al número de objetos hallados si se desea mantener en seguimiento todos. Como la vida de un objeto disminuye cuando éste no es visitado, si hay muchos puntos de atención que visitar, implicará que la vida de los objetos disminuya más rápidamente.

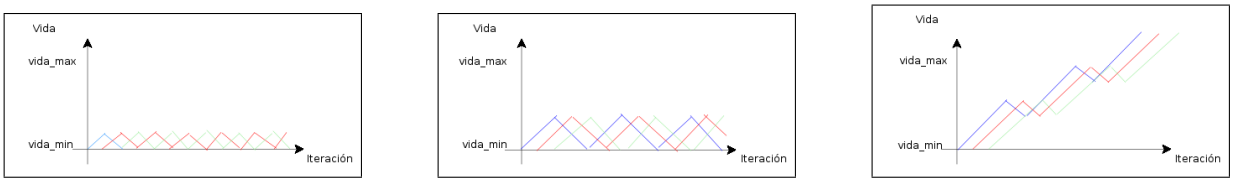


Figura 5.11: Gráfica de la vida de 3 objetos con un incremento de 1 (izquierda) de 2 (centro) y de 3 (derecha).

Señalar que la gráfica de la vida puede variar en función del momento en el que visita el objeto por primera vez. Si dos o más objetos está cercanos su curva de vida estarán siempre sobre la misma altura, en contra, si están lejanos, las curvas estarán a distinta altura (figura 5.12).

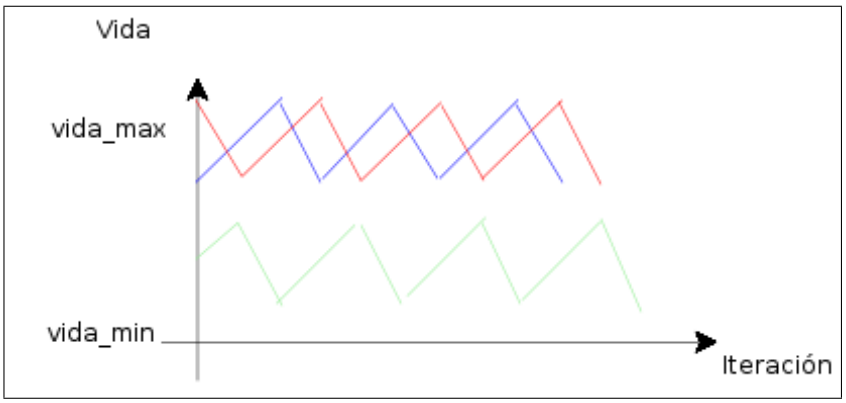


Figura 5.12: Gráfica de la vida de 3 objetos estando unade ellos lejano a los otros

También se ha observado que si dos o más objetos son identificados en la misma imagen y por consiguiente en la misma iteración, ambos poseerán la misma curva de vida.

Por último, se comprobó como el sistema era capaz de olvidar un objeto (Figura 5.14). El punto de atención correspondiente a ese objeto seguirá siendo visitado hasta que la curva de la vida de ese objeto no pase por debajo del umbral mínimo 5.15, momento en el cuál, será olvidado y, como se puede observar en la gráfica de la saliencia 5.15, la curva del objeto se cortará en el momento en el que el objeto es olvidado. En cuanto a la imagen de escena resultante (figura 5.13), la imagen correspondiente a la zona en donde se encontraba el objeto irá difuminándose al no ser visitada más. La secuencia de visita sería 1-2-3-4-5-6 hasta que olvida el objeto y ya quedaría 7-8-9-10.

Figura 5.13: Imagen de escena en donde se olvida un objeto

5.3. Experimentos con varios objetos de diferente color

En esta prueba se pretende comprobar como el sistema realiza un seguimiento sobre cuatro objetos. Dos de ellos son de color rosa y los otros dos de color azul. Para

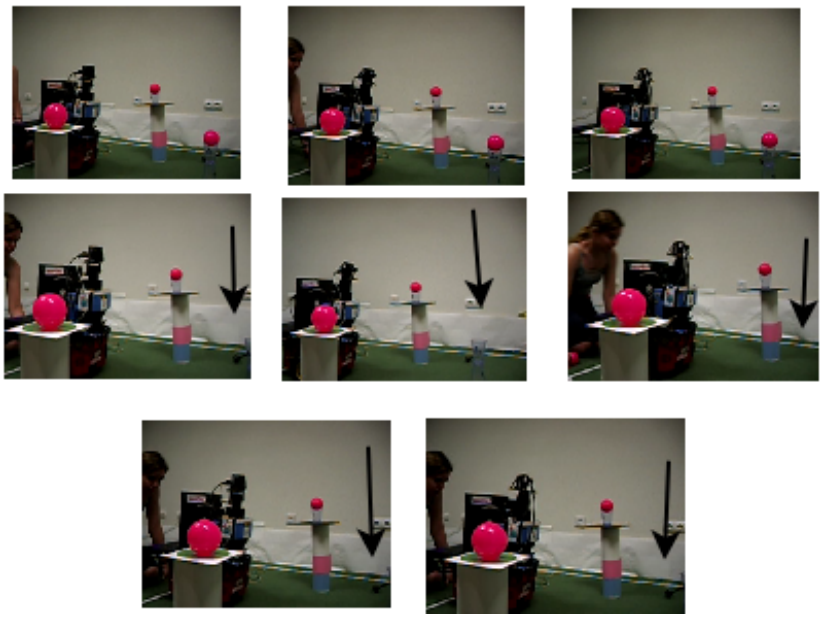


Figura 5.14: Olvido de un objeto, la posición en donde estaba el objeto se muestra con una flecha.

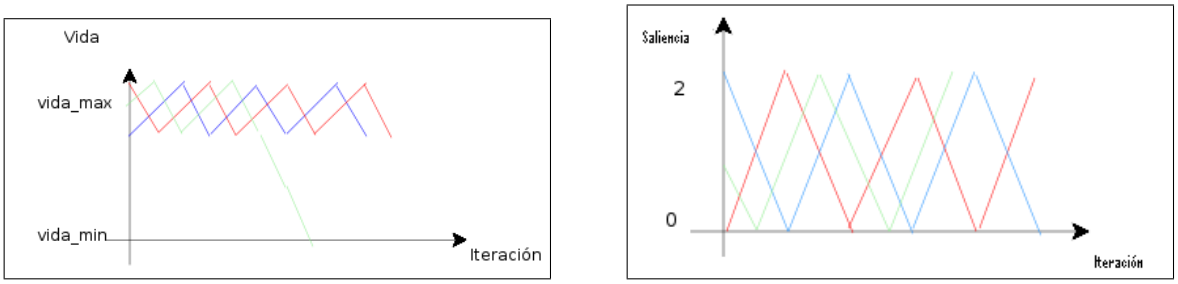


Figura 5.15: Gráfica de la vida (izquierda) y saliencia (derecha) de 3 objetos en la que se va olvidando un objeto

ello se comprobará la búsqueda de objetos relevantes, su alternancia de seguimiento y el priorizar objetos.

Al igual que en las anteriores pruebas, se comienza explorando la escena para la búsqueda de objetos. Por el mismo motivo que en la anterior, claridad en las gráficas y en los vídeos, se ha decidido bajar la probabilidad de exploración a cero en la interfaz una vez encontrado los objetos. La imagen de escena resultante una vez eliminados los puntos de exploración es la reflejada en la figura 5.16.

En un principio los cuatro objetos tienen igual prioridad, por lo que el cuello alternará por cada uno de ellos igual número de veces (figura 5.17). La combinación de poses observando los números de la imagen de escena 5.16 es 1-2-3-4-1-2-3-4 ...

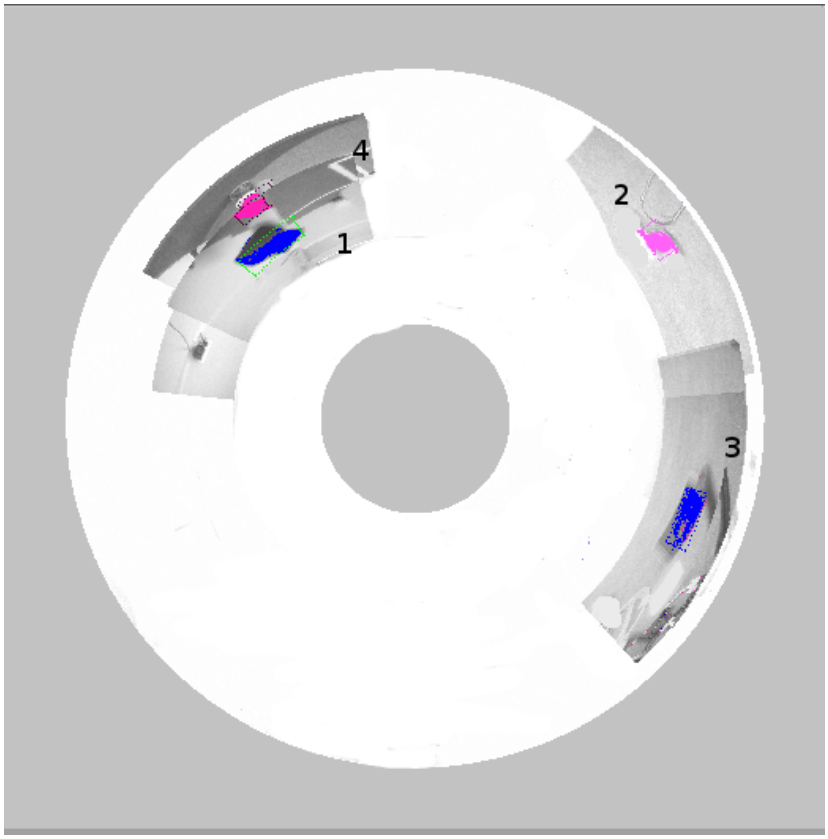


Figura 5.16: Imagen de escena de cuatro objetos con diferentes colores.

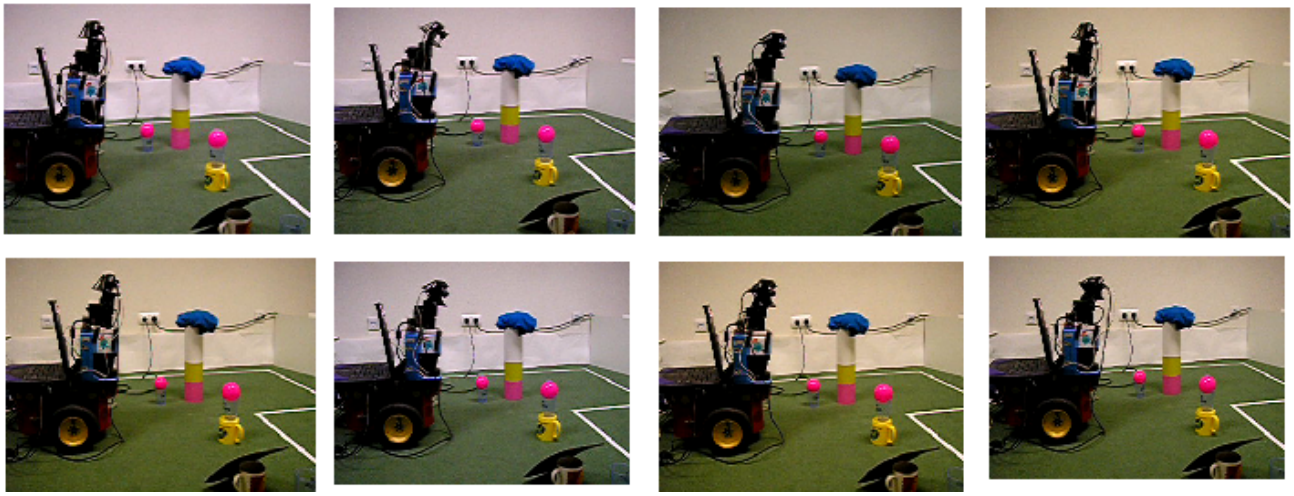


Figura 5.17: Seguimiento de 4 objetos de diferentes colores

En este caso el comportamiento en las dos dinámicas de vida y saliencia es el mismo que en el caso de la prueba anterior, ambas gráficas resultan similares que las obtenidas en la prueba 2 para objetos de igual color.

Ahora bien, cuando interesa que los objetos rosas tengan mayor prioridad que

los objetos azules, esto es, que el cuello se pose más veces sobre los objetos rosas que sobre los azules, las gráficas de vida y de saliencia variarán. Pero más especialmente la de saliencia.

Y es que los puntos de atención de los objetos rosas, como se puede ver en la figura 5.18, poseen mayor saliencia en más iteraciones que los de los azules. Esto es así porque la saliencia inicial que se le da a los puntos de atención correspondientes a los objetos con mayor prioridad difiere de la saliencia inicial normal. La combinación de poses,siguiendo la imagen de escena (figura 5.16), es 1-2-4-2-4-3-1-2-4-2-4-3-1-2-4-2-4-3.

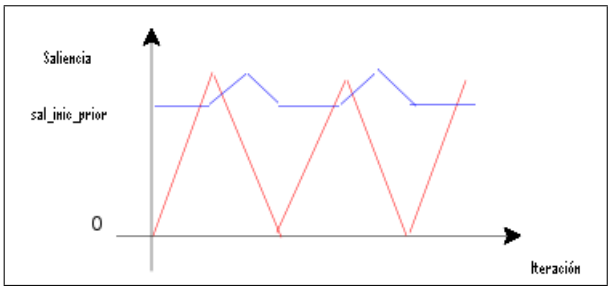


Figura 5.18: Saliencia de los puntos de atención de dos objetos, siendo un objeto más prioritario que el otro

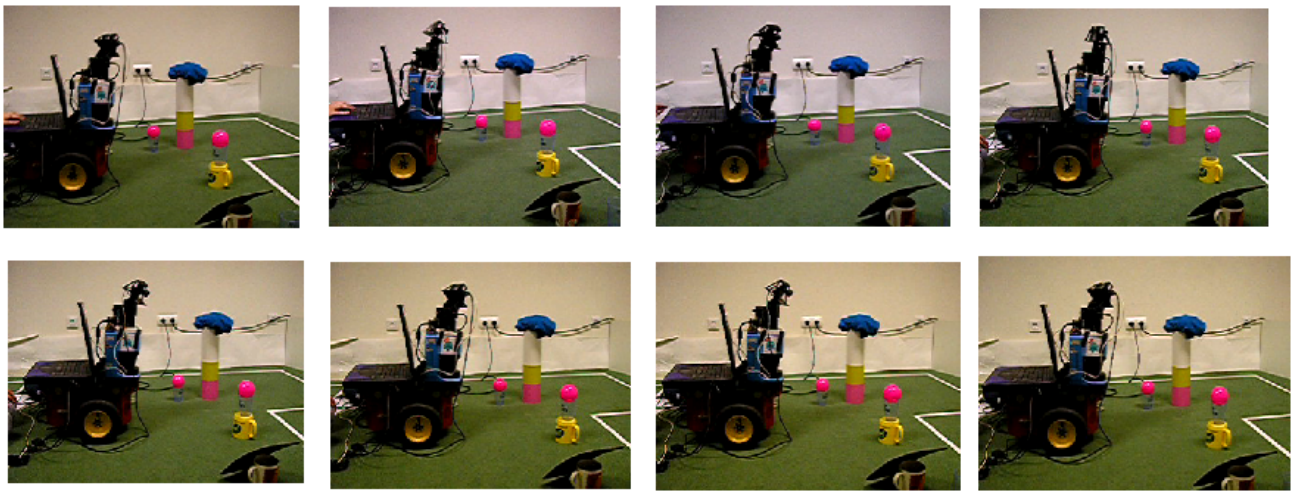


Figura 5.19: Seguimiento de 4 objetos de diferentes colores con el color rosa como prioritario

En cuanto a la vida de estos objetos prioritarios, su curva y su valor medio aumentará más al ser más visitados, en contra la de los objetos azules, sufrirá un

descenso, ya que se decrementará su vida más frecuentemente al ser menos visitados. El aumento en vida de los objetos prioritarios es directamente proporcional a su prioridad.

Capítulo 6

Conclusiones y Mejoras

En anteriores capítulos se explicó el objetivo principal de este proyecto. Tras ello, se describió el diseño y la implementación de la solución a ese problema y las pruebas que validan este sistema. Por ello, en este capítulo se presentarán las conclusiones obtenidas, la dificultades principales encontradas y las posibles mejoras para un funcionamiento más eficiente.

6.1. Conclusiones

En primer lugar resaltar que se ha conseguido el objetivo principal de este proyecto. Este objetivo consistía en realizar un algoritmo mediante el cuál, se pudiera realizar un seguimiento sobre diferentes objetos relevantes en una escena. En este sentido, se debían captar nuevos objetos, realizar un seguimiento sobre ellos y olvidarlos una vez que no estuvieran en la escena. Para su realización se empleó la plataforma descrita en el capítulo 3. Gracias a esta plataforma, el sistema puede seguir objetos dispersos en la escena, cosa que no se lograría si sólo se dispusiera de una cámara sin el cuello mecánico. Y es que con el cuello se tiene una representación de escena y con la cámara sólo se tiene información dentro del campo de visión de la misma.

De igual modo, para conseguir el objetivo principal, se implementaron dos dinámicas de atención, saliencia y vida, explicadas en el capítulo 4. La saliencia, que representa el grado de atención que tiene un punto, aumenta en cada iteración y disminuye cuando el punto es visitado. En cambio, la vida disminuye de un objeto disminuye cuando no es visitado y aumenta cuando lo es. Es sorprendente que con estas dinámicas tan simples se haya avanzado bastante en el problema de la atención sobre objetos.

El cumplimiento del objetivo principal implicaba que se cumplieran cada uno de los subobjetivos en los que se dividía. Estos eran:

1. **Percepción Monocular**, se debía conseguir que una cámara capturara imágenes continuamente para conocer la información de su entorno, y tras ello, analizar dichas imágenes para identificar los objetos relevantes. La captura se ha conseguido empleando la plataforma *JDE* (sección 3.4), ya que dicha plataforma daba resuelto el problema de la captura de imágenes. Para la identificación de los objetos relevantes se ha realizado un filtro y una segmentación. Como dichos objetos son relevantes por su color, dicho filtro será un filtro de color en el espacio *RGB*. La segmentación será una basada en un histograma con doble umbral puesto que con ello se consiguen buenos resultados y es rápido. La visualización de la imagen monocular, una vez filtrada y segmentada, se hace empleando el esquema `guiscene` (capítulo 4).

2. **Representación en Escena**, se debía tener una representación visual de la escena. Esto se ha conseguido empleando un cuello mecánico sobre el que se sitúa una cámara. Este cuello permitía el acceso a zonas en las que la cámara, debido a su limitación en cuanto a ángulo de visión, no llegaba. El cuello realiza un recorrido a lo largo de la escena. La cámara en cada uno de los puntos en los que el cuello se posa toma una imagen. Estas imágenes forman la imagen de escena (sección 4.3). Esto es similar al movimiento sacádico que realizan los ojos en los humanos.

3. **Control de Atención**, se debía realizar un control de atención sobre los diferentes objetos identificados. Para ello se han usado dos dinámicas, vida y saliencia, descritas en la sección 4.4.1. Estas dinámicas nos permiten saber en todo momento a qué punto debe dirigirse el cuello y tener una idea exacta de los objetos que hay en la escena y dónde están. Cabe señalar que ha sido sorprendente el ver que con unas dinámicas de atención tan simples, se ha llegado a avanzar en un problema tan grande como es el de atención.

Además de los subobjetivos, se han cumplido los requisitos funcionales descritos en el capítulo 2. Era esencial que se cumplieran ya que no hubiera sido aceptable un sistema que no cumpliera sus propios requisitos. La manera en la que se han resuelto es la siguiente:

1. La identificación de los objetos en una imagen se ha llevado a cabo siguiendo diferentes etapas (sección 4.2). Esto es, la captura de imágenes, un filtrado de las mismas para resaltar los objetos relevantes, una segmentación para aislarlos y una posterior identificación de los objetos. Todo ello se ha logrado al implementar una biblioteca en *C*, para facilitar su utilización en posteriores proyectos de semejantes características.

2. Se ha logrado alternar sobre los diferentes objetos relevantes que aparecen en la escena. Este problema se ha conseguido implementando la dinámica de saliencia (sección 4.4.1). Esta dinámica nos indica en todo momento qué puntos hay que atender. Asímismo, ajustando determinados parámetros de la misma, se ha conseguido introducir un nuevo parámetro, la prioridad (sección 4.4.3) que poseen objetos con semejantes características. En cada momento, zonas con determinadas características pueden resultar más interesantes y actractivas que otras.

3. Se ha logrado olvidar objetos no observados. Este problema se ha resuelto implementando la dinámica de vida (sección 4.4.1). Esta dinámica permite en cada momento saber qué objetos aún tienen interés y cuáles no.

4. Se ha logrado realizar barridos de exploración para buscar nuevos objetos de interés (sección 4.4.2). Para ello se han introducido puntos aleatorios y puntos que siguen una determinada trayectoria.

Señalar, que el diseño se ha concebido siguiendo la arquitectura *JDE* (sección 3.4). Se han diseñado dos esquemas, uno de ellos, el `guiscene`, para la ayuda en la visualización de los datos obtenidos en el sistema (imagen monocular, imagen de escena, ...) y en la depuración del mismo. Para ello se ha desarrollado una interfaz mediante la biblioteca *XForms*. Desde este esquema se puede activar y desactivar el segundo esquema, el `scene`. Este segundo esquema es para el control de atención. Este es un esquema de percepción, pero no pasivo sino activo. Ya que a partir de la información obtenida por la cámara se moverá el cuello para mantener un refresco vivo de la escena. En este esquema, a su vez, se han implementado las dos dinámicas descritas, la dinámica de vida y la de saliencia, para el control de atención.

La captura de imágenes y su tratamiento se ha logrado en un tiempo real y de manera vivaz. El filtro de color empleado para distinguir los objetos relevantes ha sido realizado en el espacio *RGB*. Se podría a ver empleado un filtro en el espacio *HSI*, pero los resultados obtenidos al usar el primer filtrado han sido bastante rápidos, óptimos

y discriminantes, como se ha podido ver en las pruebas realizadas en el capítulo 5. Señalar también que el sistema se ha comportado de manera robusta frente a cambios en la luminosidad (luz natural, luz artificial, ...).

El sistema ha sido validado experimentalmente como se muestra en el capítulo 5. Se han hecho diferentes pruebas y se ha comprobado en ellas como es capaz de identificar objetos, seguirlos cuando estos se mueven, alternar entre ellos, distinguir cuáles son los nuevos y cúales son los existentes, priorizarlos según su color y olvidarlos cuando han desaparecido.

Finalmente, señalar que los resultados de este proyecto, el código fuente de los programas desarrollados, y vídeos demostración, están disponibles en la web del grupo[1].

6.2. Líneas Futuras

Tras explicar las conclusiones obtenidas en este proyecto, se resume a continuación varias de las líneas identificadas por las que se puede mejorar y extender el sistema realizado.

Una de las posibles mejoras de este proyecto es en vez de seguir objetos por color, realizar un nuevo filtro para localizar y seguir objetos con otras características diferentes del color. Por ejemplo atender objetos según su luminosidad, su tamaño, su forma, etc. Un buen ejemplo sería el de seguir caras. Esto podría facilitar su futura utilización en cámaras de vigilancia. Se podría identificar nuevas personas y realizar un seguimiento sobre ellas y una vez que no estuvieran, olvidarlas.

Otra mejora sería el usar dos cámaras en lugar de una. Ahora mismo la atención se realiza en *2D*, pero al incorporar una nueva cámara se podría realizar en *3D* y se añadiría una nueva perspectiva, la distancia. No sólo se sabría la posición *(pan, tilt)* o *(latitud, longitud)* que tienen los objetos, sino que además se sabría a qué distancia se encuentran.

Otra posible mejora sería emplear el filtro de *Kalman* para realizar una es-

[1] http://gsyc.escet.urjc.es/robotica

timación de velocidad. Hasta ahora se le indica al cuello mecánico que se pose en la posición en donde se encontró un objeto anteriormente. Con el filtro de *Kalman*, se podría estimar la posición en donde ese objeto podría estar cuando se le vuelva a visitar. De este modo se realizaría un seguimiento más correcto. Actualmente, si un objeto visitado se va moviendo, cuando se le vuelva a visitar no va a estar en la posición anterior y es posible que se le olvide. Con el filtro de Kalman no se iría a la posición anterior, sino que se estimaría su posición actual en función de la trayectoria seguida y sería probable que el cuello mecánico se posara en la posición más cercana a la posición actual, con ello mejorará la calidad del seguimiento.

Bibliografía

[Cañas03] Cañas, J. M.: *"Jerarquía Dinámica de Esquemas para la generación de comportamiento autónomo"*, Tesis Doctoral, Universidad Politécnica de Madrid, 2003.

[Cañas04] Cañas, J.M., Matellán, V., Montúfar, R.: *"Programación de robots móviles"*, Artículo para RIAI, Revista Igeroamericana de Automática e Informática Industrial, 2004.

[Cañas05] Cañas, J.M., Martínez de la Casa Puebla, M.: *"Overt visual attention inside JDE control architecture"*, Artículo para EPIA, 2005.

[Gomez02] Gómez Gómez, V. M. : *"Comportamiento sigue pared en un robot con visión local"*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2002.

[SanMartin02] San Martín, F.: *"Comportamiento sigue pelota en un robot con visión local."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2002.

[Martinez03] Martínez de la Casa Puebla, M.: *"Comportamiento sigue pelota con visión cenital."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2003.

[Crespo03] Crespo, M. A.: *"Localización probabilística en un robot con visión local."*, Proyecto Fin de Carrera, Universidad Politécnica de Madrid, 2003.

[Lopez05] López Fernández, A.: *"Localización visual del robot Pioneer."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2005.

[Isado05] Isado, J. R.: *"Navegación global utilizando método del gradiente."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2005.

[Lopez05] López Romero, A.: *"Navegación global utilizando grafo de visibilidad."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2005.

[Pantilt03] *"Computer Controlled Pan-Tilt Unit."*, User's Manual, 2003

[CañasP04] Cañas Plaza, J. M.: *"Manual de programación de robots con JDE."*, Manual Técnico, Universidad Rey Juan Carlos, 2004